

MuPDFCore

Generated by Doxygen 1.9.4

1 MuPDFCore: Multiplatform .NET bindings for MuPDF	1
1.1 Getting started	1
1.2 Usage	2
1.2.1 Documentation	2
1.2.2 Examples	2
1.2.3 MuPDFCore library	2
1.2.4 Structured text representation	5
1.2.5 Optical Character Recognition (OCR) using Tesseract	5
1.2.6 MuPDFCore.MuPDFRenderer control	6
1.3 Building from source	6
1.3.1 1. Building libmupdf	7
1.3.1.1 Tips for compiling MuPDF 1.20.0:	7
1.3.2 2. Building MuPDFWrapper	10
1.3.2.1 Windows (x86 and x64)	10
1.3.2.2 Windows (arm64)	10
1.3.2.3 macOS and Linux	11
1.3.3 3. Creating the MuPDFCore NuGet package	11
1.3.4 4. Running tests	11
1.3.4.1 Windows	12
1.3.4.2 macOS and Linux	12
1.4 Note about MuPDFCore and .NET Framework 	13
2 Namespace Index	15
2.1 Package List	15
3 Hierarchical Index	17
3.1 Class Hierarchy	17
4 Class Index	19
4.1 Class List	19
5 File Index	21
5.1 File List	21
6 Namespace Documentation	23
6.1 Avalonia Namespace Reference	23
6.2 Avalonia.Animation Namespace Reference	23
6.3 MuPDFCore Namespace Reference	23
6.3.1 Enumeration Type Documentation	25
6.3.1.1 DocumentOutputFileTypes	25
6.3.1.2 DocumentRestrictions	25
6.3.1.3 EncryptionState	26
6.3.1.4 ExitCodes	26
6.3.1.5 InputFileTypes	27

6.3.1.6 PasswordTypes	27
6.3.1.7 PixelFormats	28
6.3.1.8 RasterOutputFileTypes	28
6.3.1.9 RestrictionState	28
6.4 MuPDFCore.MuPDFRenderer Namespace Reference	29
7 Class Documentation	31
7.1 MuPDFCore.DisposableIntPtr Class Reference	31
7.1.1 Detailed Description	31
7.1.2 Constructor & Destructor Documentation	32
7.1.2.1 DisposableIntPtr() [1/2]	32
7.1.2.2 DisposableIntPtr() [2/2]	32
7.1.3 Member Function Documentation	32
7.1.3.1 Dispose()	32
7.2 MuPDFCore.DocumentLockedException Class Reference	33
7.2.1 Detailed Description	33
7.3 MuPDFCore.MessageEventArgs Class Reference	33
7.3.1 Detailed Description	34
7.3.2 Constructor & Destructor Documentation	34
7.3.2.1 MessageEventArgs()	34
7.3.3 Property Documentation	34
7.3.3.1 Message	34
7.4 MuPDFCore.MuPDF Class Reference	35
7.4.1 Detailed Description	35
7.4.2 Member Function Documentation	35
7.4.2.1 RedirectOutput()	35
7.4.2.2 ResetOutput()	36
7.4.3 Event Documentation	36
7.4.3.1 StandardErrorMessage	36
7.4.3.2 StandardOutputMessage	36
7.5 MuPDFCore.MuPDFContext Class Reference	36
7.5.1 Detailed Description	37
7.5.2 Constructor & Destructor Documentation	37
7.5.2.1 MuPDFContext()	37
7.5.3 Member Function Documentation	37
7.5.3.1 ClearStore()	38
7.5.3.2 Dispose()	38
7.5.3.3 ShrinkStore()	38
7.5.4 Property Documentation	38
7.5.4.1 StoreMaxSize	38
7.5.4.2 StoreSize	39
7.6 MuPDFCore.MuPDFDocument Class Reference	39

7.6.1 Detailed Description	41
7.6.2 Constructor & Destructor Documentation	41
7.6.2.1 MuPDFDocument() [1/5]	42
7.6.2.2 MuPDFDocument() [2/5]	42
7.6.2.3 MuPDFDocument() [3/5]	42
7.6.2.4 MuPDFDocument() [4/5]	43
7.6.2.5 MuPDFDocument() [5/5]	43
7.6.3 Member Function Documentation	44
7.6.3.1 ClearCache()	44
7.6.3.2 CreateDocument() [1/2]	44
7.6.3.3 CreateDocument() [2/2]	44
7.6.3.4 Dispose()	46
7.6.3.5 ExtractText() [1/2]	46
7.6.3.6 ExtractText() [2/2]	47
7.6.3.7 ExtractTextAsync()	47
7.6.3.8 GetMultiThreadedRenderer()	48
7.6.3.9 GetRenderedSize() [1/2]	48
7.6.3.10 GetRenderedSize() [2/2]	49
7.6.3.11 GetStructuredTextPage() [1/2]	49
7.6.3.12 GetStructuredTextPage() [2/2]	50
7.6.3.13 GetStructuredTextPageAsync()	50
7.6.3.14 Render() [1/6]	51
7.6.3.15 Render() [2/6]	52
7.6.3.16 Render() [3/6]	52
7.6.3.17 Render() [4/6]	53
7.6.3.18 Render() [5/6]	53
7.6.3.19 Render() [6/6]	54
7.6.3.20 SaveImage() [1/2]	54
7.6.3.21 SaveImage() [2/2]	55
7.6.3.22 TryUnlock() [1/2]	55
7.6.3.23 TryUnlock() [2/2]	56
7.6.3.24 WriteImage() [1/2]	56
7.6.3.25 WriteImage() [2/2]	57
7.6.4 Property Documentation	57
7.6.4.1 ClipToPageBounds	57
7.6.4.2 EncryptionState	57
7.6.4.3 Pages	58
7.6.4.4 Restrictions	58
7.6.4.5 RestrictionState	58
7.7 MuPDFCore.MuPDFException Class Reference	58
7.7.1 Detailed Description	59
7.7.2 Member Data Documentation	59

7.7.2.1 ErrorCode	59
7.8 MuPDFCore.MuPDFImageStructuredTextBlock Class Reference	59
7.8.1 Detailed Description	60
7.8.2 Member Function Documentation	60
7.8.2.1 GetEnumerator()	60
7.8.3 Property Documentation	60
7.8.3.1 Count	60
7.8.3.2 this[int index]	61
7.8.3.3 Type	61
7.9 MuPDFCore.MuPDFMultiThreadedPageRenderer Class Reference	61
7.9.1 Detailed Description	62
7.9.2 Member Function Documentation	62
7.9.2.1 Abort()	62
7.9.2.2 Dispose()	62
7.9.2.3 GetProgress()	62
7.9.2.4 GetSpanItem()	62
7.9.2.5 Render() [1/2]	63
7.9.2.6 Render() [2/2]	63
7.9.3 Property Documentation	64
7.9.3.1 ThreadCount	64
7.10 MuPDFCore.MuPDFPage Class Reference	64
7.10.1 Detailed Description	65
7.10.2 Member Function Documentation	65
7.10.2.1 Dispose()	65
7.10.3 Property Documentation	65
7.10.3.1 Bounds	65
7.10.3.2 PageNumber	66
7.11 MuPDFCore.MuPDFPageCollection Class Reference	66
7.11.1 Detailed Description	67
7.11.2 Member Function Documentation	67
7.11.2.1 Dispose()	67
7.11.2.2 GetEnumerator()	67
7.11.3 Property Documentation	67
7.11.3.1 Count	67
7.11.3.2 Length	67
7.11.3.3 this[int index]	67
7.12 MuPDFCore.MuPDFStructuredTextAddress Struct Reference	68
7.12.1 Detailed Description	69
7.12.2 Constructor & Destructor Documentation	69
7.12.2.1 MuPDFStructuredTextAddress()	69
7.12.3 Member Function Documentation	70
7.12.3.1 CompareTo()	70

7.12.3.2 Equals() [1/2]	70
7.12.3.3 Equals() [2/2]	71
7.12.3.4 GetHashCode()	71
7.12.3.5 Increment()	71
7.12.3.6 operator"!="()	71
7.12.3.7 operator<()	72
7.12.3.8 operator<=()	72
7.12.3.9 operator==()	73
7.12.3.10 operator>()	73
7.12.3.11 operator>=()	73
7.12.4 Member Data Documentation	74
7.12.4.1 BlockIndex	74
7.12.4.2 CharacterIndex	74
7.12.4.3 LineIndex	74
7.13 MuPDFCore.MuPDFStructuredTextAddressSpan Class Reference	75
7.13.1 Detailed Description	75
7.13.2 Constructor & Destructor Documentation	75
7.13.2.1 MuPDFStructuredTextAddressSpan()	75
7.13.3 Member Data Documentation	75
7.13.3.1 End	76
7.13.3.2 Start	76
7.14 MuPDFCore.MuPDFStructuredTextBlock Class Reference	76
7.14.1 Detailed Description	77
7.14.2 Member Enumeration Documentation	77
7.14.2.1 Types	77
7.14.3 Member Function Documentation	77
7.14.3.1 GetEnumerator()	77
7.14.4 Property Documentation	78
7.14.4.1 BoundingBox	78
7.14.4.2 Count	78
7.14.4.3 this[int index]	78
7.14.4.4 Type	78
7.15 MuPDFCore.MuPDFStructuredTextCharacter Class Reference	79
7.15.1 Detailed Description	79
7.15.2 Member Function Documentation	79
7.15.2.1 ToString()	79
7.15.3 Property Documentation	80
7.15.3.1 BoundingQuad	80
7.15.3.2 Character	80
7.15.3.3 CodePoint	80
7.15.3.4 Color	80
7.15.3.5 Origin	80

7.15.3.6 Size	81
7.16 MuPDFCore.MuPDFStructuredTextLine Class Reference	81
7.16.1 Detailed Description	82
7.16.2 Member Enumeration Documentation	82
7.16.2.1 WritingModes	82
7.16.3 Member Function Documentation	82
7.16.3.1 GetEnumerator()	83
7.16.3.2 ToString()	83
7.16.4 Property Documentation	83
7.16.4.1 BoundingBox	83
7.16.4.2 Characters	83
7.16.4.3 Count	84
7.16.4.4 Direction	84
7.16.4.5 Text	84
7.16.4.6 this[int index]	84
7.16.4.7 WritingMode	85
7.17 MuPDFCore.MuPDFStructuredTextPage Class Reference	85
7.17.1 Detailed Description	86
7.17.2 Member Function Documentation	86
7.17.2.1 GetClosestHitAddress()	86
7.17.2.2 GetEnumerator()	86
7.17.2.3 GetHighlightQuads()	87
7.17.2.4 GetHitAddress()	87
7.17.2.5 GetText()	87
7.17.2.6 Search()	88
7.17.3 Property Documentation	88
7.17.3.1 Count	88
7.17.3.2 StructuredTextBlocks	89
7.17.3.3 this[int index]	89
7.17.3.4 this[MuPDFStructuredTextAddress address]	89
7.18 MuPDFCore.MuPDFTextStructuredTextBlock Class Reference	90
7.18.1 Detailed Description	90
7.18.2 Member Function Documentation	91
7.18.2.1 GetEnumerator()	91
7.18.2.2 ToString()	91
7.18.3 Property Documentation	91
7.18.3.1 Count	91
7.18.3.2 Lines	91
7.18.3.3 this[int index]	92
7.18.3.4 Type	92
7.19 MuPDFCore.OCRProgressInfo Class Reference	92
7.19.1 Detailed Description	92

7.19.2 Property Documentation	92
7.19.2.1 Progress	92
7.20 MuPDFCore.MuPDFRenderer.PDFRenderer Class Reference	93
7.20.1 Detailed Description	96
7.20.2 Member Enumeration Documentation	96
7.20.2.1 PointerEventHandlers	96
7.20.3 Constructor & Destructor Documentation	97
7.20.3.1 PDFRenderer()	97
7.20.4 Member Function Documentation	97
7.20.4.1 Contain()	97
7.20.4.2 Cover()	97
7.20.4.3 GetProgress()	97
7.20.4.4 GetSelectedText()	98
7.20.4.5 Initialize() [1/4]	98
7.20.4.6 Initialize() [2/4]	99
7.20.4.7 Initialize() [3/4]	99
7.20.4.8 Initialize() [4/4]	100
7.20.4.9 InitializeAsync() [1/4]	100
7.20.4.10 InitializeAsync() [2/4]	101
7.20.4.11 InitializeAsync() [3/4]	102
7.20.4.12 InitializeAsync() [4/4]	103
7.20.4.13 ReleaseResources()	103
7.20.4.14 Render()	104
7.20.4.15 Search()	104
7.20.4.16 SelectAll()	104
7.20.4.17 SetDisplayAreaNow()	104
7.20.4.18 ZoomStep()	105
7.20.5 Member Data Documentation	105
7.20.5.1 BackgroundProperty	105
7.20.5.2 DisplayAreaProperty	105
7.20.5.3 HighlightBrushProperty	106
7.20.5.4 HighlightedRegionsProperty	106
7.20.5.5 IsViewerInitializedProperty	106
7.20.5.6 PageBackgroundProperty	106
7.20.5.7 PageNumberProperty	107
7.20.5.8 PageSizeProperty	107
7.20.5.9 PointerEventHandlerTypeProperty	107
7.20.5.10 RenderThreadCountProperty	107
7.20.5.11 SelectionBrushProperty	108
7.20.5.12 SelectionProperty	108
7.20.5.13 ZoomEnabledProperty	108
7.20.5.14 ZoomIncrementProperty	108

7.20.5.15 ZoomProperty	109
7.20.6 Property Documentation	109
7.20.6.1 Background	109
7.20.6.2 DisplayArea	109
7.20.6.3 HighlightBrush	109
7.20.6.4 HighlightedRegions	110
7.20.6.5 IsViewerInitialized	110
7.20.6.6 PageBackground	110
7.20.6.7 PageNumber	110
7.20.6.8 PageSize	110
7.20.6.9 PointerEventHandlersType	111
7.20.6.10 RenderThreadCount	111
7.20.6.11 Selection	111
7.20.6.12 SelectionBrush	111
7.20.6.13 Zoom	111
7.20.6.14 ZoomEnabled	112
7.20.6.15 ZoomIncrement	112
7.21 MuPDFCore.PointF Struct Reference	112
7.21.1 Detailed Description	112
7.21.2 Constructor & Destructor Documentation	112
7.21.2.1 PointF()	112
7.21.3 Member Data Documentation	113
7.21.3.1 X	113
7.21.3.2 Y	113
7.22 MuPDFCore.Quad Struct Reference	113
7.22.1 Detailed Description	114
7.22.2 Constructor & Destructor Documentation	114
7.22.2.1 Quad()	114
7.22.3 Member Function Documentation	114
7.22.3.1 Contains()	115
7.22.4 Member Data Documentation	115
7.22.4.1 LowerLeft	115
7.22.4.2 LowerRight	115
7.22.4.3 UpperLeft	115
7.22.4.4 UpperRight	116
7.23 MuPDFCore.Rectangle Struct Reference	116
7.23.1 Detailed Description	117
7.23.2 Constructor & Destructor Documentation	117
7.23.2.1 Rectangle() [1/2]	117
7.23.2.2 Rectangle() [2/2]	117
7.23.3 Member Function Documentation	118
7.23.3.1 Contains() [1/2]	118

7.23.3.2 Contains() [2/2]	118
7.23.3.3 Intersect()	119
7.23.3.4 Round() [1/2]	119
7.23.3.5 Round() [2/2]	119
7.23.3.6 Split()	120
7.23.3.7 ToQuad()	120
7.23.4 Member Data Documentation	120
7.23.4.1 X0	121
7.23.4.2 X1	121
7.23.4.3 Y0	121
7.23.4.4 Y1	121
7.23.5 Property Documentation	121
7.23.5.1 Height	121
7.23.5.2 Width	122
7.24 Avalonia.Animation.RectTransition Class Reference	122
7.24.1 Detailed Description	122
7.24.2 Member Function Documentation	122
7.24.2.1 DoTransition()	123
7.25 MuPDFCore.RenderProgress Class Reference	123
7.25.1 Detailed Description	123
7.25.2 Property Documentation	123
7.25.2.1 ThreadRenderProgresses	123
7.26 MuPDFCore.RoundedRectangle Struct Reference	124
7.26.1 Detailed Description	124
7.26.2 Constructor & Destructor Documentation	124
7.26.2.1 RoundedRectangle()	124
7.26.3 Member Function Documentation	125
7.26.3.1 Split()	125
7.26.4 Member Data Documentation	125
7.26.4.1 X0	125
7.26.4.2 X1	126
7.26.4.3 Y0	126
7.26.4.4 Y1	126
7.26.5 Property Documentation	126
7.26.5.1 Height	126
7.26.5.2 Width	126
7.27 MuPDFCore.RoundedSize Struct Reference	127
7.27.1 Detailed Description	127
7.27.2 Constructor & Destructor Documentation	127
7.27.2.1 RoundedSize()	127
7.27.3 Member Function Documentation	128
7.27.3.1 Split()	128

7.27.4 Member Data Documentation	128
7.27.4.1 Height	128
7.27.4.2 Width	128
7.28 MuPDFCore.Size Struct Reference	129
7.28.1 Detailed Description	129
7.28.2 Constructor & Destructor Documentation	129
7.28.2.1 Size() [1/2]	129
7.28.2.2 Size() [2/2]	130
7.28.3 Member Function Documentation	130
7.28.3.1 Split()	130
7.28.4 Member Data Documentation	130
7.28.4.1 Height	130
7.28.4.2 Width	131
7.29 MuPDFCore.TesseractLanguage Class Reference	131
7.29.1 Detailed Description	133
7.29.2 Member Enumeration Documentation	133
7.29.2.1 Best	133
7.29.2.2 BestScripts	136
7.29.2.3 Fast	137
7.29.2.4 FastScripts	140
7.29.3 Constructor & Destructor Documentation	141
7.29.3.1 TesseractLanguage() [1/6]	142
7.29.3.2 TesseractLanguage() [2/6]	142
7.29.3.3 TesseractLanguage() [3/6]	142
7.29.3.4 TesseractLanguage() [4/6]	143
7.29.3.5 TesseractLanguage() [5/6]	143
7.29.3.6 TesseractLanguage() [6/6]	143
7.29.4 Property Documentation	144
7.29.4.1 Language	144
7.29.4.2 Prefix	144
7.30 MuPDFCore.RenderProgress.ThreadRenderProgress Struct Reference	144
7.30.1 Detailed Description	145
7.30.2 Member Data Documentation	145
7.30.2.1 MaxProgress	145
7.30.2.2 Progress	145
8 File Documentation	147
8.1 PDFRenderer.cs	147
8.2 PDFRenderer.Properties.cs	167
8.3 RectTransition.cs	171
8.4 MuPDF.cs	172
8.5 MuPDFContext.cs	191

8.6 MuPDFDisplayList.cs	193
8.7 MuPDFDocument.cs	194
8.8 MuPDFMultiThreadedPageRenderer.cs	214
8.9 MuPDFPage.cs	222
8.10 MuPDFStructuredTextPage.cs	224
8.11 Rectangles.cs	241
8.12 TesseractLanguage.cs	249
8.13 Utils.cs	269
Index	273

Chapter 1

MuPDFCore: Multiplatform .NET bindings for MuPDF

MuPDFCore is a set of multiplatform .NET bindings for [MuPDF](#). It can render PDF, XPS, EPUB and other formats to raster images returned either as raw bytes, or as image files in multiple formats (including PNG and PSD). It also supports multithreading.

It also includes **MuPDFCore.MuPDFRenderer**, an [Avalonia](#) control to display documents compatible with [MuPDFCore](#) in [Avalonia](#) windows (with multithreaded rendering).

The library is released under the [AGPLv3](#) licence.

1.1 Getting started

The [MuPDFCore](#) library targets .NET Standard 2.0, thus it can be used in projects that target .NET Standard 2.0+, .NET Core 2.0+, .NET 5.0+, .NET Framework 4.6.1 (note) and possibly others. [MuPDFCore](#) includes a pre-compiled native library, which currently supports the following platforms:

- Windows x86 (32 bit)
- Windows x64 (64 bit)
- Windows arm64 (ARM 64 bit)
- Linux x64 (64 bit)
- Linux arm64/aarch64 (ARM 64 bit)
- macOS Intel x86_64 (64 bit)
- macOS Apple silicon (ARM 64 bit)

To use the library in your project, you should install the [MuPDFCore NuGet package](#) and/or the [MuPDFCore.PDFRenderer NuGet package](#). When you publish a program that uses [MuPDFCore](#), the correct native library for the target architecture will automatically be copied to the build folder (but see the note for .NET Framework).

Note: you should make sure that end users on Windows install the [Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, 2019 and 2022](#) for their platform, otherwise they will get an error message stating that `MuPDFWrapper.dll` could not be loaded because a module was not found.

1.2 Usage

1.2.1 Documentation

Interactive documentation for the library can be accessed from the [documentation website](#). A [PDF reference manual](#) is also available.

1.2.2 Examples

The `Demo` folder in the repository contains some examples of how the library can be used to extract pages from a PDF or XPS document, render them to a raster image, or combine them in a new document

The `PDFViewerDemo` folder contains a complete (though minimal) example of a PDF viewer program built around the `MuPDFCore.MuPDFRenderer.PDFRenderer` control.

Note that these examples intentionally avoid any error handling code: in a production setting, you should typically make sure that calls to `MuPDFCore` library functions are within a `try...catch` block to handle any resulting `MuPDFExceptions`.

1.2.3 MuPDFCore library

The first step when using `MuPDFCore` is to create a `MuPDFCore.MuPDFContext` object that is used internally by the MuPDF library to store various things:

```
MuPDFContext context = new MuPDFContext();
```

This object is `IDisposable`, therefore you should always call the `Dispose()` method on it once you are done with it (or, better yet, wrap it in a `using` directive). In most instances, you will only need one instance of `MuPDFContext` for your whole application.

Amongst other things, MuPDF uses this context to store a cache of "assets" (e.g. images or fonts) that have been used while rendering documents and that may be needed in future. This requires some memory: by default, the maximum size of this cache store is 256MB; however, if you want to restrict how much memory can be used, you can alter this by providing a `long` value to constructor, indicating the size in bites for the store. A value of 0 means that the store can grow up to an unlimited size. Furthermore, you can clear the cache completely by using the `MuPDFContext.ClearCache` method, or partially by using the `MuPDFContext.ShrinkCache` method.

Once you have obtained a `MuPDFContext`, you can use it to open a `MuPDFDocument`. A document can be opened from a file on disk:

```
MuPDFDocument document = new MuPDFDocument(context, "path/to/file");
```

Or from a `byte[]` array (in this case, you will have to specify the format of the document):

```
byte[] data;
...
MuPDFDocument document = new MuPDFDocument(context, data, InputFileTypes.PDF);
```

Or from a `MemoryStream` (in this case too, you will have to specify the format of the document):

```
MemoryStream stream;
...
MuPDFDocument document = new MuPDFDocument(context, ref stream, InputFileTypes.PDF);
```

The `MemoryStream` is passed with the `ref` keyword to indicate that the `MuPDFDocument` will take care of appropriately disposing it once it finishes using it.

A `MuPDFDocument` is also `IDisposable` and should be properly disposed of to avoid memory leaks.

Important note: the constructor taking a `byte[]` and the one taking a `MemoryStream` will not copy the data bytes before sending them to the native MuPDF library functions. Rather, they will *pin them in place*. This is a **bad thing** because it will mess up with the Garbage Collector's management of memory. Therefore, this is only suitable for short-lived objects. If you need to initialise a long-lived document object from memory, you should first copy the data to unmanaged memory and then use one of the constructors that take an `IntPtr` parameter, e.g.:

```
byte[] data;
...
//Allocate enough unmanaged memory
IntPtr ptr = Marshal.AllocHGlobal(data.Length);
//Copy the byte array to unmanaged memory
Marshal.Copy(data, 0, ptr, data.Length);
//Wrap the pointer in an IDisposable
IDisposable dispIntPtr = new DisposableIntPtr(ptr);
//Create the document
MuPDFDocument document = new MuPDFDocument(ctx, ptr, data.Length, InputFileTypes.PDF, ref dispIntPtr);
```

The `DisposableIntPtr` class is a wrapper around a pointer that calls `Marshal.FreeHGlobal` on it once it is disposed. Passing it as the final optional parameter of `MuPDFDocument` constructor (again by reference, to indicate that the document takes ownership of the object) makes sure that the memory is properly freed once the document is disposed.

After having obtained a document, you can do many things with it: for example, you can render a page and save the results to a file on disk, or you can collect multiple pages and combine them in a new document. Code to do this can be found in the `Program.cs` file of the Demo project.

Furthermore, you can render a page directly to memory:

```
byte[] pixelData = document.Render(0, 1, PixelFormats.RGBA);
```

This method renders page 0 (i.e. the first page of the document) at a 1x resolution (1pt in the document is equivalent to 1px in the image), preserving alpha (transparency) information, and returns the image as an array of the bytes that constitute the pixel data (four bytes per pixel). A variation of this method allows you to supply a rectangular region of the page that you would like to render, rather than the whole page.

Alternatively, if you already know where the image data should be put (e.g. because you are using some kind of graphics library that lets you manipulate the pixel data of its images), you can use the methods that take an `IntPtr` destination:

```
IntPtr destination;
...
document.Render(0, 1, PixelFormats.RGBA, destination);
```

In this case, **you have to make sure that there is enough memory to hold the resulting image!** Otherwise, an `AccessViolationException` will occur and your program will usually fail catastrophically. Since it may sometimes be hard to determine how much memory a particular image will need (especially because of subtle differences in the rounding routines, which can cause images to be 1px larger or shorter than expected), the `GetRenderedSize` method is provided, which returns the number of bytes that will be needed to render a certain page. For example:

```
//Get the number of bytes that will be necessary to hold the rendered page at the given resolution.
int sizeInBytes = document.GetRenderedSize(0, 1, PixelFormats.RGBA);
//Allocate an appropriate amount of memory.
IntPtr destination = Marshal.AllocHGlobal(sizeInBytes);
//Again, we use a DisposableIntPtr to make sure that we are freeing the memory when we are done with it.
using (DisposableIntPtr holder = new DisposableIntPtr(destination))
{
    //Make sure that all the parameters match those of the call to GetRenderedSize, or the size of the
    //resulting image may be different than expected! Even a translation of 1px could have catastrophic
    //consequences.
    document.Render(0, 1, PixelFormats.RGBA, destination);
}
```

Finally, **none of these methods are inherently thread-safe!** E.g. you cannot render multiple pages of the same document (nor multiple regions of a single page) by simply performing multiple calls to `MuPDFDocument.Render` in parallel. For multi-threaded operation, you must instead use a `MuPDFMultiThreadedPage.Render`. You can obtain one from a document:

```
MuPDFMultiThreadedPageRenderer renderer = document.GetMultiThreadedRenderer(0, 2);
```

This method obtains an object that can be used to render the first page of the document using two threads. By using the `Render` method of this object, the page can be rendered. The page will be rendered to a number of separate tiles equal to the number of threads, which will then be your responsibility to appropriately "stitch up" (e.g. if you

want to display them on screen, you could just place them appropriately). The size of each tile (and the position it should occupy) can be computed by using the `Split` method of the `RoundedSize` struct.

Furthermore, multiple `MuPDFMultiThreadedPageRenderers` can be used in parallel, which makes it possible e.g. to render every page in the document at the same time (while also using multiple threads to render each page). The following example will render all the pages in a document at the same time in RGBA format at a 1.5x zoom, using 2 threads for each page:

```
//Create a MuPDFContext with a using statement, so that it gets disposed at the right time.
using MuPDFContext context = new MuPDFContext();
//Open the document also with a using statement.
using MuPDFDocument document = new MuPDFDocument(context, "path/to/file.pdf");
//Create arrays to hold the objects for the various pages
//Renderers: one per page
MuPDFMultiThreadedPageRenderer[] renderers = new MuPDFMultiThreadedPageRenderer[document.Pages.Count];
//Page size: one per page
RoundedSize[] renderedPageSizes = new RoundedSize[document.Pages.Count];
//Boundaries of the tiles that make up each page: one array per page, with one element per thread
RoundedRectangle[][] tileBounds = new RoundedRectangle[document.Pages.Count][][];
//Addresses of the memory areas where the image data of the tiles will be stored: one array per page, with
//one element per thread
IntPtr[][] destinations = new IntPtr[document.Pages.Count][];
//Cycle through the pages in the document to initialise everything
for (int i = 0; i < document.Pages.Count; i++)
{
    //Initialise the renderer for the current page, using two threads (total number of threads: number of
    //pages x 2
    renderers[i] = document.GetMultiThreadedRenderer(i, 2);
    //Determine the boundaries of the page when it is rendered with a 1.5x zoom factor
    RoundedRectangle roundedBounds = document.Pages[i].Bounds.Round(1.5);
    renderedPageSizes[i] = new RoundedSize(roundedBounds.Width, roundedBounds.Height);
    //Determine the boundaries of each tile by splitting the total size of the page by the number of
    //threads.
    tileBounds[i] = renderedPageSizes[i].Split(renderers[i].ThreadCount);
    destinations[i] = new IntPtr[renderers[i].ThreadCount];
    for (int j = 0; j < renderers[i].ThreadCount; j++)
    {
        //Allocate the required memory for the j-th tile of the i-th page.
        //Since we will be rendering with a 24-bit-per-pixel format, the required memory in bytes is height
        //x width x 3.
        destinations[i][j] = Marshal.AllocHGlobal(tileBounds[i][j].Height * tileBounds[i][j].Width * 3);
    }
}
//Start the actual rendering operations in parallel.
Parallel.For(0, document.Pages.Count, i =>
{
    renderers[i].Render(renderedPageSizes[i], document.Pages[i].Bounds, destinations[i], PixelFormats.RGB);
});
//The code in this for-loop is not really part of MuPDFCore - it just shows an example of using VectSharp to
// "stitch" the tiles up and produce the full image.
for (int i = 0; i < document.Pages.Count; i++)
{
    //Create a new (empty) image to hold the whole page.
    VectSharp.Page renderedPage = new VectSharp.Page(renderedPageSizes[i].Width,
        renderedPageSizes[i].Height);
    //Draw each tile onto the image.
    for (int j = 0; j < renderers[i].ThreadCount; j++)
    {
        //Create a raster image object containing the pixel data. Yay, we do not need to copy/marshal
        //anything!
        VectSharp.RasterImage tile = new VectSharp.RasterImage(destinations[i][j], tileBounds[i][j].Width,
            tileBounds[i][j].Height, false, false);
        //Draw the tile on the main image page.
        renderedPage.Graphics.DrawRasterImage(tileBounds[i][j].X0, tileBounds[i][j].Y0, tile);
    }
    //Save the full page as a PNG image.
    renderedPage.SaveAsPNG("page" + i.ToString() + ".png");
}
//Clean-up code.
for (int i = 0; i < document.Pages.Count; i++)
{
    //Release the allocated memory.
    for (int j = 0; j < renderers[i].ThreadCount; j++)
    {
        Marshal.FreeHGlobal(destinations[i][j]);
    }
    //Release the renderer (if you skip this, the quiescent renderer's threads will not be stopped, and your
    //application will never exit!
    renderers[i].Dispose();
}
```

1.2.4 Structured text representation

The `GetStructuredTextPage` method of the `MuPDFDocument` class makes it possible to obtain a "structured text" representation of each page of the document. This consists of a `MuPDFStructuredTextPage` object, which is a collection of 0 or more `MuPDFStructuredTextBlocks`.

Each `MuPDFStructuredTextBlock` either represents an image or a block of text, typically a paragraph (though there is no guarantee that this is the case). `MuPDFStructuredTextBlocks` are themselves collections of `MuPDFStructuredTextLines`, and each line is a collection of `MuPDFStructuredTextCharacters` (in the case of a block representing an image, it will contain a single line with a single character).

`MuPDFStructuredTextBlocks` and `MuPDFStructuredTextLines` have a `BoundingBox` property that defines a rectangle (in page units) that bounds the contents of the block/line in the page. Similarly, `MuPDFStructuredTextCharacters` have a `BoundingQuad` (rather than being a `Rectangle`, this is a `Quad`, i.e. a quadrilateral defined by its four vertices, which may or may not be a rectangle). These can be used e.g. to highlight regions of text in the page.

The `MuPDFStructuredTextPage` also has methods to determine which character contains or is closest to a specified point (useful, for example, to determine on which character the user clicked), to obtain a list of shapes that encompass a specified range of text, and to perform text searches using regular expressions.

The order of the blocks in the page (which affects the definition of a "range" of text and search operations) is the same as returned by the underlying MuPDF library, which is taken from the order the text is drawn in the source file, so may not be accurate. They can be reordered using the `Array.Sort` method on the `StructuredTextBlocks` array contained in the `MuPDFStructuredTextPage` (lines within blocks and characters within lines can be likewise reordered).

1.2.5 Optical Character Recognition (OCR) using Tesseract

MuPDF 1.18+ (embedded in `MuPDFCore` 1.3.0+) adds support for OCR using the `Tesseract` library. To access this feature in `MuPDFCore`, you can use one of the overloads of `GetStructuredTextPage` that takes a `TesseractLanguage` argument specifying the language to use for the OCR. This will run the OCR and return a `MuPDFStructuredTextPage` containing the character information obtained by Tesseract, which can be used normally. Depending on the model being used, the OCR step can take a relatively long time; therefore, the `MuPDFDocument` class also implements a `GetStructuredTextPageAsync` method, which does the same thing in an asynchronous way. The `GetStructuredTextPageAsync` method also has optional parameters to report the OCR progress and to make it possible to cancel its execution.

Objects of the `TesseractLanguage` class contain information used to locate the trained language model file that is used by Tesseract. Normally, when using Tesseract, you would have to ensure that the trained language model files are available on the user's computer; however, this class implements some "clever" logic to download the necessary files on demand.

In general, MuPDF provides Tesseract with a "language name" (e.g. "eng"). Tesseract then looks for a file called `eng.traineddata` either in the folder specified by the `TESSDATA_PREFIX` environment variable, or, if the variable is not defined, in a subfolder of the current working directory called `tessdata`. `MuPDFCore` manipulates the value of `TESSDATA_PREFIX` (at the process level) and the language name in order to specify the language file.

The `TesseractLanguage` class has multiple constructors:

- `TesseractLanguage(string prefix, string language)`: this constructor is used to directly specify the value of `TESSDATA_PREFIX` and the language name. The library does not process these in any way. If `prefix` is null, the value of `TESSDATA_PREFIX` is not changed, and Tesseract uses the system value.

- `TesseractLanguage(string fileName)`: with this constructor, you can directly specify the path to a trained language model file. You can obtain such a file from [the tessdata_fast repository](#) or from [the tessdata_best repository](#). If the file does not have a `.traineddata` extension, it will be copied in a temporary location.
- `TesseractLanguage(Fast language, bool useAnyCached = false)` \ `TesseractLanguage(FastScript language, bool useAnyCached = false)` \ `TesseractLanguage(Best language, bool useAnyCached = false)`

With these constructors, you can specify a language from the list of available languages defined in the `TesseractLanguage.Fast`, `TesseractLanguage.FastScript`, `TesseractLanguage.Best`, and `TesseractLanguage.BestScript` enums.

[MuPDFCore](#) will then look for the trained model file corresponding to the selected language, relative to the *path of the executable*, in a folder called `tessdata/fast` and then in a folder called `fast` (or `best`, depending on the overload; for the overloads taking a script name, it looks in `tessdata/fast/script` or `fast/script` instead).

If the language file is not found in either of these folders, it then looks for it in a subfolder called `tessdata/fast` in `Environment.SpecialFolder.LocalApplicationData`. If the optional argument `useAnyCached` is `true`, it also looks for the language file in the same folder as the executable, and then in the `best` (or `fast`) subfolders. In this case, for example, if the language file for `TesseractLanguage.Fast.Eng` is not available, but the file for `TesseractLanguage.Best.Eng` is available, the latter will be used.

Finally, if the language file could not be found in any of the possible paths, [MuPDFCore](#) will download it from the appropriate repository and place it in the appropriate subfolder of the `tessdata` folder in `Environment.SpecialFolder.LocalApplicationData`. The file will then be reused as necessary.

The `TESSDATA_PREFIX` and language name will then be set accordingly to where the file was located.

This means that if you use one of these constructors you do not have to worry about the language files being installed in the right place; as long as the user has an Internet connection, the library will download the language files as necessary.

1.2.6 MuPDFCore.MuPDFRenderer control

To use the `PDFRenderer` control in an [Avalonia](#) application, first of all you need to add it to your [Avalonia](#) Window, e.g. in the XAML:

```
<Window xmlns="https://github.com/avaloniaui"
    ...
    xmlns:mupdf="clr-namespace:MuPDFCore.MuPDFRenderer;assembly=MuPDFCore.MuPDFRenderer"
    Opened="WindowOpened"
    ... >
    <mupdf:PDFRenderer Name="MuPDFRenderer" />
</Window>
```

You then need to initialise it from the backing code, e.g. in a `WindowOpened` event:

```
private void WindowOpened(object sender, EventArgs e)
{
    this.FindControl<PDFRenderer>("MuPDFRenderer").Initialize("path/to/file.pdf");
}
```

This way, the renderer will start showing the first page of the specified document, using a number of rendering threads that is decided based on the number of processors in the computer. There are many other ways to initialise a `PDFRenderer`, so make sure to look at the [documentation](#) to see the other possibilities!

1.3 Building from source

Building the [MuPDFCore](#) library from source requires the following steps:

1. Building the `libmupdf` native library
2. Building the `MuPDFWrapper` native library
3. Creating the `MuPDFCore` library NuGet package

Steps 1 and 2 need to be performed on all of Windows, macOS and Linux, and on the various possible architectures (x86, x64 and arm64 for Windows, x64/Intel and arm64/Apple for macOS, and x64 and arm64 for Linux - no cross-compiling)! Otherwise, some native assets will be missing and it will not be possible to build the NuGet package.

1.3.1 1. Building `libmupdf`

You can download the open-source (GNU AGPL) MuPDF source code from [here](#). You will need to uncompress the source file and compile the library on Windows, macOS and Linux. You need the following files:

- From Windows (x86, x64, arm64):
 - `libmupdf.lib`
- From macOS (Intel - x64, Apple silicon - arm64):
 - `libmupdf.a`
 - `libmupdf-third.a`
- From Linux (x64, arm64):
 - `libmupdf.a`
 - `libmupdf-third.a`

Note that the files from macOS and Linux are different, despite sharing the same name.

For convenience, these compiled files for MuPDF 1.19.1 are included in the `native/MuPDFWrapper/lib folder` of this repository.

1.3.1.1 Tips for compiling MuPDF 1.20.0:

- On all platforms:
 - You do not need to follow the instructions in `thirdparty/tesseract.txt`, as in this version the `leptonica` and `tesseract` libraries are already included in the source archive.
 - Delete or comment line 316 in `source/fitz/output.c` (the `fz_throw` invocation within the `buffer_seek` method - this should leave the `buffer_seek` method empty). This line throws an exception when a seek operation on a buffer is attempted. The problem is that this makes it impossible to render a document as a PSD image in memory, because the `fz_write_pixmap_as_psd` method performs a few seek operations. By removing this line, we turn buffer seeks into no-ops; this doesn't seem to have catastrophic side-effects and the PSD documents produced in this way appear to be fine.
- On Windows (x64):
 - Open the `platform/win32/mupdf.sln` solution in Visual Studio. You should get a prompt to retarget your projects. Accept the default settings (latest Windows SDK and v143 of the tools).
 - Select the `ReleaseTesseract` configuration and `x64` architecture. Select every project in the solution except `javaviewer` and `javaviewerlib` and right-click to open the project properties. Go to `C/C++ > Code Generation` and set the Runtime Library to Multi-threaded DLL (`/MD`). Save everything (`CTRL+SHIFT+S`) and close Visual Studio.

- Now, open the x64 Native Tools Command Prompt for VS, move to the folder with the solution file, and build it using `msbuild mupdf.sln`
- Then, build again using `msbuild mupdf.sln /p:Configuration=Release`. Ignore the compilation errors.
- Finally, build again using `msbuild mupdf.sln /p:Configuration=ReleaseTesseract`.
- This may still show some errors, but should produce the `libmupdf.lib` file that is required in the `x64/ReleaseTesseract` folder (the file should be ~441MB in size).
- On Windows (x86):
 - You will have to use Visual Studio 2019, as Visual Studio 2022 is not supported on x86 platforms.
 - Open the `platform/win32/mupdf.sln` solution in Visual Studio and select the `Release` ← `Tesseract` configuration and Win32 architecture. Select every project in the solution except `javaviewer` and `javaviewerlib` and right-click to open the project properties. Go to `C/C++` > `Code Generation` and set the Runtime Library to Multi-threaded DLL (`/MD`). Save everything (CTRL+SHIFT+S) and close Visual Studio.
 - Now, open the x86 Native Tools Command Prompt for VS, move to the folder with the solution file, and build it using `msbuild mupdf.sln /p:Platform=Win32`
 - Then, build again using `msbuild mupdf.sln /p:Configuration=Release /p:Platform=Win32`. Ignore the compilation errors.
 - Finally, build again using `msbuild mupdf.sln /p:Configuration=ReleaseTesseract /p:Platform=Win32`.
 - This may still show some errors, but should produce the `libmupdf.lib` file that is required in the `ReleaseTesseract` folder (the file should be ~362MB in size).
- On Windows (arm64)

This is going to be a bit more complicated, because it appears that MuPDF is not meant to be built on ARM. These instructions will assume that you are building MuPDF on an ARM machine.

First of all, make sure that you have installed Visual Studio 2022 and have selected the C++ ARM64 build tools component of the "Desktop development with C++" workload.

Note: When you install Visual Studio on an ARM machine, it will complain that this is not supported and will be slow. Ignore that warning.

 - Download and extract the MuPDF source code and follow the instructions for all platforms above.
 - Add `|| defined(_M_ARM64)` at the end of line 16 in `scripts/tesseract/endianess.h`.
 - Now we need to edit a few files in the `thirdparty/tesseract/src/arch` folder.
 - * Comment or delete lines 149-177 (inclusive) in `simddetect.cpp`. You should now have an empty block between `# elif defined(_WIN32)` and `#else`. Also comment or delete lines 198-220 (inclusive) and 237-260 (inclusive).
 - * Comment or delete lines 20-22 (inclusive) in `dotproductsse.cpp`. Replace the whole body of the `DotProductSSE` method (lines 30-76) with `return DotProductNative(u, v, n);`.
 - * Comment or delete lines 20-21 (inclusive) in `dotproductavx.cpp`. Replace the whole body of the `DotProductAVX` method (lines 29-54) with `return DotProductNative(u, v, n);`.
 - * Comment or delete lines 20-21 (inclusive) in `dotproductfma.cpp`. Replace the whole body of the `DotProductFMA` method (lines 29-52) with `return DotProductNative(u, v, n);`.
 - * Delete the contents of `thirdparty/tesseract/src/arch/intsimdmatrixavx2.cpp` and `thirdparty/tesseract/src/arch/intsimdmatrixsse.cpp` (do not delete the files, just their contents).
 - * Comment or delete lines 120-121 (inclusive) in `intsimdmatrix.h`
 - Open the `platform/win32/mupdf.sln` solution in Visual Studio. You should get a prompt to retarget your projects. Accept the default settings (latest Windows SDK and v143 of the tools).

- In Visual Studio, click on the "Configuration Manager" item from the "Build" menu. In the new window, click on the drop down menu for the "Active solution platform" and select <New...>. In this new dialog, select the ARM64 platform and choose to copy the settings from x64. Leave the Create new project platforms option enabled and click on OK (this may take some time).
 - Close the Configuration Manager and select the ReleaseTesseract configuration and ARM64 architecture. Select every project in the solution except javaviewer and javaviewerlib and right-click to open the project properties. Go to C/C++ > Code Generation and set the Runtime Library to Multi-threaded DLL (/MD).
 - Open the properties for the libpkcs7 project, go to C/C++ > Preprocessor and remove HAVE←_LIBCRYPTO from the Preprocessor Definitions. Then go to Librarian > General and remove libcrypto.lib from the Additional Dependencies.
 - Save everything (CTRL+SHIFT+S) and close Visual Studio.
 - Create a new folder platform/win32/Release. Now, the problem is that the bin2coff script included with MuPDF cannot create obj files for ARM64 (only for x86 and x64). Since I could not find a version that can do this, I translated the source code of bin2coff to C# and added this option myself. You can download an ARM64 bin2coff.exe from [here](#); place it in the Release folder that you have just created.
 - Open the Developer Command Prompt for VS, move to the folder with the solution file (platform/win32), and build it using msbuild mupdf.sln /p:Configuration=Release←Tesseract. Some compilation errors may occur towards the end, but they should not matter.
 - After a while, this should produce libmupdf.lib in the ARM64/ReleaseTesseract folder (the file should be ~444MB in size).
- On Linux (x64):
 - Edit the Makefile, adding the -fPIC compiler option at the end of line 24 (which specifies the CFLAGS).
 - Make sure that you are using a recent enough version of GCC (version 7.3.1 seems to be enough).
 - Compile by running USE_TESSERACT=yes make HAVE_X11=no HAVE_GLUT=no (this builds just the command-line libraries and tools, and enables OCR through the included Tesseract library).
 - On Linux (arm64):
 - Edit the Makefile, adding the -fPIC compiler option at the end of line 24 (which specifies the CFLAGS).
 - Delete or comment line 218 in thirdparty/tesseract/src/arch/simddetect.cpp.
 - Make sure that you are using a recent enough version of GCC (version 7.3.1 seems to be enough).
 - Compile by running USE_TESSERACT=yes make HAVE_X11=no HAVE_GLUT=no (this builds just the command-line libraries and tools, and enables OCR through the included Tesseract library).
 - On macOS (Intel - x64):
 - Edit the Makefile, adding the -fPIC compiler option at the end of line 24 (which specifies the CFLAGS). Also add the -std=c++11 option at the end of line 58 (which specifies the CXX_CMD).
 - Compile by running USE_TESSERACT=yes make (this enables OCR through the included Tesseract library).
 - On macOS (Apple silicon - arm64)
 - Edit the Makefile, adding the -fPIC compiler options at the end of line 24 (which specifies the CFLAGS). Also add the -std=c++11 option at the end of line 58 (which specifies the CXX_CMD).
 - Delete or comment line 218 in thirdparty/tesseract/src/arch/simddetect.cpp.
 - Compile by running USE_TESSERACT=yes make (this enables OCR through the included Tesseract library).

1.3.2 2. Building MuPDFWrapper

Once you have the required static library files, you should download the [MuPDFCore](#) source code: [MuPDFCore-1.4.0.tar.gz](#) (or clone the repository) and place the library files in the appropriate subdirectories in the native/MuPDFWrapper/lib/ folder.

To compile MuPDFWrapper you will need [CMake](#) (version 3.8 or higher) and (on Windows) [Ninja](#).

On Windows, the easiest way to get all the required tools is probably to install [Visual Studio](#). By selecting the "Desktop development with C++" workload you should get everything you need.

On macOS, you will need to install at least the Command-Line Tools for Xcode (if necessary, you should be prompted to do this while you perform the following steps) and CMake.

Once you have everything at the ready, you will have to build MuPDFWrapper on the seven platforms.

1.3.2.1 Windows (x86 and x64)

1.

Assuming you have installed Visual Studio, you should open the "x64 Native Tools Command Prompt for VS" or the "x86 Native Tools Command Prompt for VS" (you should be able to find these in the Start menu). Take care to open the version corresponding to the architecture you are building for, otherwise you will not be able to compile the library. A normal command prompt will not work, either.

Note 1: you **must** build the library on two separate systems, one running a 32-bit version of Windows and the other running a 64-bit version. If you try to build the x86 library on an x64 system, the system will probably build a 64-bit library and place it in the 32-bit output folder, which will just make things very confusing.

Note 2 for Windows x86: for some reason, Visual Studio might install the 64-bit version of CMake and Ninja, even though you are on a 32-bit machine. If this happens, you will have to manually install the 32-bit CMake and compile a 32-bit version of Ninja. You will notice if this is an issue because the 64-bit programs will refuse to run.

1. CD to the directory where you have downloaded the [MuPDFCore](#) source code.
2. CD into the native directory.
3. Type build. This will start the build.cmd batch script that will delete any previous build and compile the library.

After this finishes, you should find a file named MuPDFWrapper.dll in the native/out/build/win-x64/↔ MuPDFWrapper/ directory or in the native/out/build/win-x86/MuPDFWrapper/ directory. Leave it there.

1.3.2.2 Windows (arm64)

1. Locate the batch file that sets up the developer command prompt environment. You can do this by finding the "Developer Command Prompt for VS" link in the start menu, then clicking on Open file location, opening the properties of the link and looking at the Target. This could be e.g. C:\Program Files\Microsoft Visual Studio\2022\Preview\Common7\Tools\VsDevCmd.bat.
2. Open a normal command prompt and invoke the batch script with the -arch=arm64 -host_arch=x86 arguments (add quotes if there are spaces in the path to the batch script), e.g.:

```
"C:\Program Files\Microsoft Visual Studio\2022\Preview\Common7\Tools\VsDevCmd.bat" -arch=arm64 -host_arch=x86
```
3. CD to the directory where you have downloaded the [MuPDFCore](#) source code.
4. CD into the native directory.
5. Type build. This will start the build.cmd batch script that will delete any previous build and compile the library.

After this finishes, you should find a file named MuPDFWrapper.dll in the native/out/build/win-arm64/↔ MuPDFWrapper/ directory. Leave it there.

1.3.2.3 macOS and Linux

1. Assuming you have everything ready, open a terminal in the folder where you have downloaded the [MuPDFCore](#) source code.
2. `cd` into the native directory.
3. Type `chmod +x build.sh`.
4. Type `./build.sh`. This will delete any previous build and compile the library.

After this finishes, you should find a file named `libMuPDFWrapper.dylib` in the `native/out/build/mac-x64/` or `MuPDFWrapper/` directory (on macOS running on an Intel x64 processor) or in the `native/out/build/mac-arm64/` or `MuPDFWrapper/` directory (on macOS running on an Apple silicon arm64 processor), and a file named `libMuPDFWrapper.so` in the `native/out/build/linux-x64/MuPDFWrapper/` directory (on Linux). Leave it there.

1.3.3 3. Creating the MuPDFCore NuGet package

Once you have the `MuPDFWrapper.dll`, `libMuPDFWrapper.dylib` and `libMuPDFWrapper.so` files, make sure they are in the correct folders (`native/out/build/xxx-yyy/MuPDFWrapper/`), **all on the same machine**.

To create the [MuPDFCore](#) NuGet package, you will need the [.NET Core 2.0 SDK or higher](#) for your platform. Once you have installed it and have everything ready, open a terminal in the folder where you have downloaded the [MuPDFCore](#) source code and type:

```
cd MuPDFCore  
dotnet pack -c Release
```

This will create a NuGet package in `MuPDFCore/bin/Release`. You can install this package on your projects by adding a local NuGet source.

1.3.4 4. Running tests

To verify that everything is working correctly, you should build the [MuPDFCore](#) test suite and run it on all platforms. To build the test suite, you will need the [.NET 6 SDK or higher](#). You will also need to have enabled the [Windows Subsystem for Linux](#).

To build the test suite:

1. Make sure that you have changed the version of the [MuPDFCore](#) NuGet package so that it is higher than the latest version of [MuPDFCore](#) in the NuGet repository (you should use a pre-release suffix, e.g. `1.4.0-a1` to avoid future headaches with new versions of [MuPDFCore](#)). This is set in line 9 of the `MuPDFCore/MuPDFCore.csproj` file.
2. Add the `MuPDFCore/bin/Release` folder to your local NuGet repositories (you can do this e.g. in Visual Studio).
3. If you have not done so already, create the [MuPDFCore](#) NuGet package following step 3 above.
4. Update line 56 of the `Tests/Tests.csproj` project file so that it refers to the version of the [MuPDFCore](#) package you have just created.

These steps ensure that you are testing the right version of [MuPDFCore](#) (i.e. your freshly built copy) and not something else that may have been cached.

Now, open a windows command line in the folder where you have downloaded the [MuPDFCore](#) source code, type `BuildTests` and press `Enter`. This will create a number of files in the `Release\MuPDFCoreTests` folder, where each file is an archive containing the tests for a certain platform and architecture:

- `MuPDFCoreTests-linux-x64.tar.gz` contains the tests for Linux environments on x64 processors.
- `MuPDFCoreTests-linux-arm64.tar.gz` contains the tests for Linux environments on arm64 processors.
- `MuPDFCoreTests-mac-x64.tar.gz` contains the tests for macOS environments on Intel processors.
- `MuPDFCoreTests-mac-arm64.tar.gz` contains the tests for macOS environments on Apple silicon processors.
- `MuPDFCoreTests-win-x64.tar.gz` contains the tests for Windows environments on x64 processors.
- `MuPDFCoreTests-win-x86.tar.gz` contains the tests for Windows environments on x86 processors.

To run the tests, copy each archive to a machine running the corresponding operating system, and extract it (note: on Windows, the default zip file manager may struggle when extracting the text file with non-latin characters; you may need to manually extract this file). Then:

1.3.4.1 Windows

- Open a command prompt and `CD` into the folder where you have extracted the contents of the test archive.
- Enter the command `MuPDFCoreTestHost` (this will run the test program).

1.3.4.2 macOS and Linux

- Open a terminal and `cd` into the folder where you have extracted the contents of the test archive.
- Enter the command `chmod +x MuPDFCoreTestHost` (this will add the executable flag to the test program).
- Enter the command `./MuPDFCoreTestHost` (this will run the test program).
- On macOS, depending on your security settings, you may get a message saying `zsh: killed` when you try to run the program. To address this, you need to sign the executable, e.g. by running `codesign --timestamp --sign <certificate> MuPDFCoreTestHost`, where `<certificate>` is the name of a code signing certificate in your keychain (e.g. Developer ID Application: John Smith). After this, you can try again to run the test program with `./MuPDFCoreTestHost`.

The test suite will start; it will print the name of each test, followed by a green `Succeeded` or a red `Failed` depending on the test result. If everything went correctly, all tests should succeed.

When all the tests have been run, the program will print a summary showing how many tests have succeeded (if any) and how many have failed (if any). If any tests have failed, a list of these will be printed, and then they will be run again one at a time, waiting for a key press before running each test (this makes it easier to follow what is going on). If you wish to kill the test process early, you can do so with `CTRL+C`.

1.4 Note about MuPDFCore and .NET Framework

If you wish to use [MuPDFCore](#) in a .NET Framework project, you will need to manually copy the native MuPDFWrapper library for the platform you are using to the executable directory (this is done automatically if you target .NET/.NET core).

One way to obtain the appropriate library files is:

1. Manually download the NuGet package for [MuPDFCore](#) (click on the "Download package" link on the right).
2. Rename the .nupkg file so that it has a .zip extension.
3. Extract the zip file.
4. Within the extracted folder, the library files are in the `runtimes/xxx/native/` folder, where `xxx` is `linux-x64`, `linux-arm64`, `osx-x64`, `osx-arm64`, `win-x64`, `win-x86` or `win-arm64`, depending on the platform you are using.

Make sure you copy the appropriate file to the same folder as the executable!

Chapter 2

Namespace Index

2.1 Package List

Here are the packages with brief descriptions (if available):

Avalonia	23
Avalonia.Animation	23
MuPDFCore	23
MuPDFCore.MuPDFRenderer	29

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Control	
MuPDFCore.MuPDFRenderer.PDFRenderer	93
MuPDFCore.MuPDFRenderer.PDFRenderer	93
EventArgs	
MuPDFCore.MessageEventArgs	33
Exception	
MuPDFCore.DocumentLockedException	33
MuPDFCore.MuPDFException	58
IComparable	
MuPDFCore.MuPDFStructuredTextAddress	68
IDisposable	
MuPDFCore.DisposableIntPtr	31
MuPDFCore.MuPDFContext	36
MuPDFCore.MuPDFDocument	39
MuPDFCore.MuPDFMultiThreadedPageRenderer	61
MuPDFCore.MuPDFPage	64
MuPDFCore.MuPDFPageCollection	66
IEquatable	
MuPDFCore.MuPDFStructuredTextAddress	68
IReadOnlyList	
MuPDFCore.MuPDFPageCollection	66
MuPDFCore.MuPDFStructuredTextBlock	76
MuPDFCore.MuPDFImageStructuredTextBlock	59
MuPDFCore.MuPDFTextStructuredTextBlock	90
MuPDFCore.MuPDFStructuredTextLine	81
MuPDFCore.MuPDFStructuredTextPage	85
MuPDFCore.MuPDF	
MuPDFCore.MuPDFStructuredTextAddressSpan	75
MuPDFCore.MuPDFStructuredTextCharacter	79
MuPDFCore.OCRProgressInfo	92
MuPDFCore.PointF	112
MuPDFCore.Quad	113
MuPDFCore.Rectangle	116
MuPDFCore.RenderProgress	123
MuPDFCore.RoundedRectangle	124

MuPDFCore.RoundedSize	127
MuPDFCore.Size	129
MuPDFCore.TesseractLanguage	131
MuPDFCore.RenderProgress.ThreadRenderProgress	144
Transition	
Avalonia.Animation.RectTransition	122

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

MuPDFCore.DisposableIntPtr	An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.	31
MuPDFCore.DocumentLockedException	The exception that is thrown when an attempt is made to render an encrypted document without supplying the required password.	33
MuPDFCore.MessageEventArgs	EventArgs for the MuPDF.StandardOutputMessage and MuPDF.StandardErrorMessage events.	33
MuPDFCore.MuPDF	Contains static methods to perform setup operations.	35
MuPDFCore.MuPDFContext	A wrapper around a MuPDF context object, which contains the exception stack and the resource cache store.	36
MuPDFCore.MuPDFDocument	A wrapper over a MuPDF document object, which contains possibly multiple pages.	39
MuPDFCore.MuPDFException	The exception that is thrown when a MuPDF operation fails.	58
MuPDFCore.MuPDFImageStructuredTextBlock	Represents a block containing a single image. The block contains a single line with a single character.	59
MuPDFCore.MuPDFMultiThreadedPageRenderer	A class that holds the necessary resources to render a page of a MuPDF document using multiple threads.	61
MuPDFCore.MuPDFPage	A wrapper over a MuPDF page object, which contains information about the page's boundaries.	64
MuPDFCore.MuPDFPageCollection	A lazy collection of MuPDFPages . Each page is loaded from the document as it is requested for the first time.	66
MuPDFCore.MuPDFStructuredTextAddress	Represents the address of a particular character in a MuPDFStructuredTextPage , in terms of block index, line index and character index.	68
MuPDFCore.MuPDFStructuredTextAddressSpan	Represents a range of characters in a MuPDFStructuredTextPage .	75
MuPDFCore.MuPDFStructuredTextBlock	Represents a structured text block containing text or an image.	76
MuPDFCore.MuPDFStructuredTextCharacter	Represents a single text character.	79

MuPDFCore.MuPDFStructuredTextLine	Represents a single line of text (i.e. characters that share a common baseline).	81
MuPDFCore.MuPDFStructuredTextPage	Represents a structured representation of the text contained in a page.	85
MuPDFCore.MuPDFTextStructuredTextBlock	Represents a block containing multiple lines of text (typically a paragraph).	90
MuPDFCore.OCRProgressInfo	Describes OCR progress.	92
MuPDFCore.MuPDFRenderer.PDFRenderer	A control to render PDF documents (and other formats), potentially using multiple threads.	93
MuPDFCore.PointF	Represents a point.	112
MuPDFCore.Quad	Represents a quadrilateral (not necessarily a rectangle).	113
MuPDFCore.Rectangle	Represents a rectangle.	116
Avalonia.Animation.RectTransition	Transition class that handles AvaloniaProperty with Rect types.	122
MuPDFCore.RenderProgress	Holds a summary of the progress of the current rendering operation.	123
MuPDFCore.RoundedRectangle	Represents a rectangle using only integer numbers.	124
MuPDFCore.RoundedSize	Represents the size of a rectangle using only integer numbers.	127
MuPDFCore.Size	Represents the size of a rectangle.	129
MuPDFCore.TesseractLanguage	Represents a language used by Tesseract OCR.	131
MuPDFCore.RenderProgress.ThreadRenderProgress	Holds the progress of a single thread.	144

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

MuPDFCore.MuPDFRenderer/PDFRenderer.cs	147
MuPDFCore.MuPDFRenderer/PDFRenderer.Properties.cs	167
MuPDFCore.MuPDFRenderer/RectTransition.cs	171
MuPDFCore/MuPDF.cs	172
MuPDFCore/MuPDFContext.cs	191
MuPDFCore/MuPDFDisplayList.cs	193
MuPDFCore/MuPDFDocument.cs	194
MuPDFCore/MuPDFMultiThreadedPageRenderer.cs	214
MuPDFCore/MuPDFPage.cs	222
MuPDFCore/MuPDFStructuredTextPage.cs	224
MuPDFCore/Rectangles.cs	241
MuPDFCore/TesseractLanguage.cs	249
MuPDFCore/Utils.cs	269

Chapter 6

Namespace Documentation

6.1 Avalonia Namespace Reference

6.2 Avalonia.Animation Namespace Reference

Classes

- class [RectTransition](#)
Transition class that handles `AvaloniaProperty` with `Rect` types.

6.3 MuPDFCore Namespace Reference

Classes

- class [DisposableIntPtr](#)
An `IDisposable` wrapper around an `IntPtr` that frees the allocated memory when it is disposed.
- class [DocumentLockedException](#)
The exception that is thrown when an attempt is made to render an encrypted document without supplying the required password.
- class [MessageEventArgs](#)
EventArgs for the `MuPDF.StandardOutputMessage` and `MuPDF.StandardErrorMessage` events.
- class [MuPDF](#)
Contains static methods to perform setup operations.
- class [MuPDFContext](#)
A wrapper around a `MuPDF` context object, which contains the exception stack and the resource cache store.
- class [MuPDFDocument](#)
A wrapper over a `MuPDF` document object, which contains possibly multiple pages.
- class [MuPDFException](#)
The exception that is thrown when a `MuPDF` operation fails.
- class [MuPDFImageStructuredTextBlock](#)
Represents a block containing a single image. The block contains a single line with a single character.
- class [MuPDFMultiThreadedPageRenderer](#)
A class that holds the necessary resources to render a page of a `MuPDF` document using multiple threads.

- class [MuPDFPage](#)
A wrapper over a MuPDF page object, which contains information about the page's boundaries.
- class [MuPDFPageCollection](#)
A lazy collection of MuPDFPages. Each page is loaded from the document as it is requested for the first time.
- struct [MuPDFStructuredTextAddress](#)
Represents the address of a particular character in a MuPDFStructuredTextPage, in terms of block index, line index and character index.
- class [MuPDFStructuredTextAddressSpan](#)
Represents a range of characters in a MuPDFStructuredTextPage.
- class [MuPDFStructuredTextBlock](#)
Represents a structured text block containing text or an image.
- class [MuPDFStructuredTextCharacter](#)
Represents a single text character.
- class [MuPDFStructuredTextLine](#)
Represents a single line of text (i.e. characters that share a common baseline).
- class [MuPDFStructuredTextPage](#)
Represents a structured representation of the text contained in a page.
- class [MuPDFTextStructuredTextBlock](#)
Represents a block containing multiple lines of text (typically a paragraph).
- class [OCRProgressInfo](#)
Describes OCR progress.
- struct [PointF](#)
Represents a point.
- struct [Quad](#)
Represents a quadrilateral (not necessarily a rectangle).
- struct [Rectangle](#)
Represents a rectangle.
- class [RenderProgress](#)
Holds a summary of the progress of the current rendering operation.
- struct [RoundedRectangle](#)
Represents a rectangle using only integer numbers.
- struct [RoundedSize](#)
Represents the size of a rectangle using only integer numbers.
- struct [Size](#)
Represents the size of a rectangle.
- class [TesseractLanguage](#)
Represents a language used by Tesseract OCR.

Enumerations

- enum [ExitCodes](#) {

`ERR_CANNOT_CREATE_CONTEXT = 129 , ERR_CANNOT_REGISTER_HANDLERS = 130 ,`

`ERR_CANNOT_OPEN_FILE = 131 , ERR_CANNOT_COUNT_PAGES = 132 ,`

`ERR_CANNOT_RENDER = 134 , ERR_CANNOT_OPEN_STREAM = 135 , ERR_CANNOT_LOAD_PAGE`

`= 136 , ERR_CANNOT_COMPUTE_BOUNDS = 137 ,`

`ERR_CANNOT_INIT_MUTEX = 138 , ERR_CANNOT_CLONE_CONTEXT = 139 , ERR_CANNOT_SAVE =`

`140 , ERR_CANNOT_CREATE_BUFFER = 141 ,`

`ERR_CANNOT_CREATE_WRITER = 142 , ERR_CANNOT_CLOSE_DOCUMENT = 143 , ERR_CANNOT_CREATE_PAGE`

`= 144 , ERR_CANNOT_POPULATE_PAGE = 145 ,`

`EXIT_SUCCESS = 0 }`

Exit codes returned by native methods describing various errors that can occur.

- enum `InputFileTypes` {

 `PDF` = 0 , `XPS` = 1 , `CBZ` = 2 , `PNG` = 3 ,

 `JPEG` = 4 , `BMP` = 5 , `GIF` = 6 , `TIFF` = 7 ,

 `PNM` = 8 , `PAM` = 9 , `EPUB` = 10 , `FB2` = 11 }

 File types supported in input by the library.
- enum `RasterOutputFileTypes` { `PNM` = 0 , `PAM` = 1 , `PNG` = 2 , `PSD` = 3 }

 Raster image file types supported in output by the library.
- enum `DocumentOutputFileTypes` { `PDF` = 0 , `SVG` = 1 , `CBZ` = 2 }

 Document file types supported in output by the library.
- enum `PixelFormats` { `RGB` = 0 , `RGBA` = 1 , `BGR` = 2 , `BGRA` = 3 }

 Pixel formats supported by the library.
- enum `EncryptionState` { `Unencrypted` = 0 , `Encrypted` = 1 , `Unlocked` = 2 }

 Possible document encryption states.
- enum `RestrictionState` { `Unrestricted` = 0 , `Restricted` = 1 , `Unlocked` = 2 }

 Possible document restriction states.
- enum `DocumentRestrictions` {

 `None` = 0 , `Print` = 1 , `Copy` = 2 , `Edit` = 4 ,

 `Annotate` = 8 }

 Document restrictions.
- enum `PasswordTypes` { `None` = 0 , `User` = 1 , `Owner` = 2 }

 Password types.

6.3.1 Enumeration Type Documentation

6.3.1.1 DocumentOutputFileTypes

`enum MuPDFCore.DocumentOutputFileTypes`

Document file types supported in output by the library.

Enumerator

<code>PDF</code>	Portable Document Format.
<code>SVG</code>	Scalable Vector Graphics.
<code>CBZ</code>	Comic book archive format.

Definition at line 214 of file [MuPDF.cs](#).

6.3.1.2 DocumentRestrictions

`enum MuPDFCore.DocumentRestrictions`

Document restrictions.

Enumerator

None	No operation is restricted.
Print	Printing the document is restricted.
Copy	Copying the document is restricted.
Edit	Editing the document is restricted.
Annotate	Annotating the document is restricted.

Definition at line 303 of file [MuPDF.cs](#).

6.3.1.3 EncryptionState

enum [MuPDFCore.EncryptionState](#)

Possible document encryption states.

Enumerator

Unencrypted	The document is not encrypted.
Encrypted	The document is encrypted and a user password is necessary to render it.
Unlocked	The document is encrypted and the correct user password has been supplied.

Definition at line 261 of file [MuPDF.cs](#).

6.3.1.4 ExitCodes

enum [MuPDFCore.ExitCodes](#)

Exit codes returned by native methods describing various errors that can occur.

Enumerator

ERR_CANNOT_CREATE_CONTEXT	An error occurred while creating the context object.
ERR_CANNOT_REGISTER_HANDLERS	An error occurred while registering the default document handlers with the context.
ERR_CANNOT_OPEN_FILE	An error occurred while opening a file.
ERR_CANNOT_COUNT_PAGES	An error occurred while determining the total number of pages in the document.
ERR_CANNOT_RENDER	An error occurred while rendering the page.
ERR_CANNOT_OPEN_STREAM	An error occurred while opening the stream.
ERR_CANNOT_LOAD_PAGE	An error occurred while loading the page.
ERR_CANNOT_COMPUTE_BOUNDS	An error occurred while computing the page bounds.
ERR_CANNOT_INIT_MUTEX	An error occurred while initialising the mutexes for the lock mechanism.
ERR_CANNOT_CLONE_CONTEXT	An error occurred while cloning the context.

Enumerator

ERR_CANNOT_SAVE	An error occurred while saving the page to a raster image file.
ERR_CANNOT_CREATE_BUFFER	An error occurred while creating the output buffer.
ERR_CANNOT_CREATE_WRITER	An error occurred while creating the document writer.
ERR_CANNOT_CLOSE_DOCUMENT	An error occurred while finalising the document file.
ERR_CANNOT_CREATE_PAGE	An error occurred while creating an empty structured text page.
ERR_CANNOT_POPULATE_PAGE	An error occurred while populating the structured text page
EXIT_SUCCESS	No error occurred. All is well.

Definition at line 31 of file [MuPDF.cs](#).

6.3.1.5 InputFileTypes

```
enum MuPDFCore.InputFileTypes
```

File types supported in input by the library.

Enumerator

PDF	Portable Document Format.
XPS	XML Paper Specification document.
CBZ	Comic book archive file (ZIP archive containing page scans).
PNG	Portable Network Graphics format.
JPEG	Joint Photographic Experts Group image.
BMP	Bitmap image.
GIF	Graphics Interchange Format.
TIFF	Tagged Image File Format.
PNM	Portable aNyMap graphics format.
PAM	Portable Arbitrary Map graphics format.
EPUB	Electronic PUBLication document.
FB2	FictionBook document.

Definition at line 122 of file [MuPDF.cs](#).

6.3.1.6 PasswordTypes

```
enum MuPDFCore.PasswordTypes
```

Password types.

Enumerator

None	No password.
User	The password corresponds to the user password.
Owner	The password corresponds to the owner password.

Definition at line 334 of file [MuPDF.cs](#).

6.3.1.7 PixelFormats

`enum MuPDFCore.PixelFormats`

Pixel formats supported by the library.

Enumerator

RGB	24bpp RGB format.
RGBA	32bpp RGBA format.
BGR	24bpp BGR format.
BGRA	32bpp BGRA format.

Definition at line 235 of file [MuPDF.cs](#).

6.3.1.8 RasterOutputFileTypes

`enum MuPDFCore.RasterOutputFileTypes`

Raster image file types supported in output by the library.

Enumerator

PNM	Portable aNyMap graphics format.
PAM	Portable Arbitrary Map graphics format.
PNG	Portable Network Graphics format.
PSD	PhotoShop Document format.

Definition at line 188 of file [MuPDF.cs](#).

6.3.1.9 RestrictionState

`enum MuPDFCore.RestrictionState`

Possible document restriction states.

Enumerator

Unrestricted	The document does not have any restrictions associated to it.
Restricted	Some restrictions apply to the document. An owner password is required to remove these restrictions.
Unlocked	The document had some restrictions and the correct owner password has been supplied.

Definition at line 282 of file [MuPDF.cs](#).

6.4 MuPDFCore.MuPDFRenderer Namespace Reference

Classes

- class [PDFRenderer](#)

A control to render PDF documents (and other formats), potentially using multiple threads.

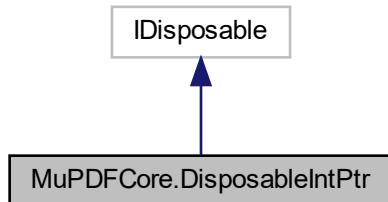
Chapter 7

Class Documentation

7.1 MuPDFCore.DisposableIntPtr Class Reference

An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.

Inheritance diagram for MuPDFCore.DisposableIntPtr:



Public Member Functions

- [DisposableIntPtr \(IntPtr pointer\)](#)
Create a new `DisposableIntPtr`.
- [DisposableIntPtr \(IntPtr pointer, long bytesAllocated\)](#)
Create a new `DisposableIntPtr`, adding memory pressure to the GC to account for the allocation of unmanaged memory.
- void [Dispose \(\)](#)

7.1.1 Detailed Description

An IDisposable wrapper around an IntPtr that frees the allocated memory when it is disposed.

Definition at line 406 of file [MuPDF.cs](#).

7.1.2 Constructor & Destructor Documentation

7.1.2.1 DisposableIntPtr() [1/2]

```
MuPDFCore.DisposableIntPtr.DisposableIntPtr (
    IntPtr pointer )
```

Create a new [DisposableIntPtr](#).

Parameters

<i>pointer</i>	The pointer that should be freed upon disposing of this object.
----------------	---

Definition at line [422](#) of file [MuPDF.cs](#).

7.1.2.2 DisposableIntPtr() [2/2]

```
MuPDFCore.DisposableIntPtr.DisposableIntPtr (
    IntPtr pointer,
    long bytesAllocated )
```

Create a new [DisposableIntPtr](#), adding memory pressure to the GC to account for the allocation of unmanaged memory.

Parameters

<i>pointer</i>	The pointer that should be freed upon disposing of this object.
<i>bytesAllocated</i>	The number of bytes that have been allocated, for adding memory pressure.

Definition at line [432](#) of file [MuPDF.cs](#).

7.1.3 Member Function Documentation

7.1.3.1 Dispose()

```
void MuPDFCore.DisposableIntPtr.Dispose ( )
```

Definition at line [468](#) of file [MuPDF.cs](#).

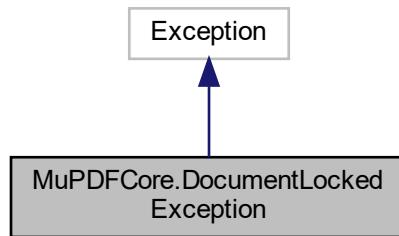
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDF.cs

7.2 MuPDFCore.DocumentLockedException Class Reference

The exception that is thrown when an attempt is made to render an encrypted document without supplying the required password.

Inheritance diagram for MuPDFCore.DocumentLockedException:



7.2.1 Detailed Description

The exception that is thrown when an attempt is made to render an encrypted document without supplying the required password.

Definition at line 494 of file [MuPDF.cs](#).

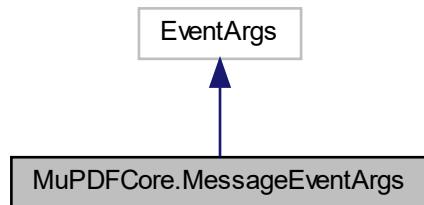
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDF.cs

7.3 MuPDFCore.MessageEventArgs Class Reference

EventArgs for the [MuPDF.StandardOutputMessage](#) and [MuPDF.StandardErrorMessage](#) events.

Inheritance diagram for MuPDFCore.MessageEventArgs:



Public Member Functions

- [MessageEventArgs](#) (string message)
Create a new [MessageEventArgs](#) instance.

Properties

- string [Message](#) [get]
The message that has been logged.

7.3.1 Detailed Description

EventArgs for the [MuPDF.StandardOutputMessage](#) and [MuPDF.StandardErrorMessage](#) events.

Definition at line 550 of file [MuPDF.cs](#).

7.3.2 Constructor & Destructor Documentation

7.3.2.1 [MessageEventArgs\(\)](#)

```
MuPDFCore.MessageEventArgs.MessageEventArgs (
    string message )
```

Create a new [MessageEventArgs](#) instance.

Parameters

<code>message</code>	The message that has been logged.
----------------------	-----------------------------------

Definition at line 561 of file [MuPDF.cs](#).

7.3.3 Property Documentation

7.3.3.1 [Message](#)

```
string MuPDFCore.MessageEventArgs.Message [get]
```

The message that has been logged.

Definition at line 555 of file [MuPDF.cs](#).

The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDF.cs](#)

7.4 MuPDFCore.MuPDF Class Reference

Contains static methods to perform setup operations.

Static Public Member Functions

- static async Task [RedirectOutput \(\)](#)

Redirects output messages from the native MuPDF library to the StandardOutputMessage and StandardErrorMessage events. Note that this has side-effects.

- static void [ResetOutput \(\)](#)

Reset the default standard output and error streams for the native MuPDF library.

Events

- static EventHandler< [MessageEventArgs](#) > [StandardOutputMessage](#)

This event is invoked when RedirectOutput has been called and the native MuPDF library writes to the standard output stream.

- static EventHandler< [MessageEventArgs](#) > [StandardErrorMessage](#)

This event is invoked when RedirectOutput has been called and the native MuPDF library writes to the standard error stream.

7.4.1 Detailed Description

Contains static methods to perform setup operations.

Definition at line 570 of file [MuPDF.cs](#).

7.4.2 Member Function Documentation

7.4.2.1 [RedirectOutput\(\)](#)

```
static async Task MuPDFCore.MuPDF.RedirectOutput ( ) [static]
```

Redirects output messages from the native MuPDF library to the StandardOutputMessage and StandardErrorMessage events. Note that this has side-effects.

Returns

A Task that finishes when the output streams have been redirected.

Definition at line 599 of file [MuPDF.cs](#).

7.4.2.2 ResetOutput()

```
static void MuPDFCore.MuPDF.ResetOutput ( ) [static]
```

Reset the default standard output and error streams for the native [MuPDF](#) library.

Definition at line 819 of file [MuPDF.cs](#).

7.4.3 Event Documentation

7.4.3.1 StandardErrorMessage

```
EventHandler<MessageEventArgs> MuPDFCore.MuPDF.StandardErrorMessage [static]
```

This event is invoked when [RedirectOutput](#) has been called and the native [MuPDF](#) library writes to the standard error stream.

Definition at line 593 of file [MuPDF.cs](#).

7.4.3.2 StandardOutputMessage

```
EventHandler<MessageEventArgs> MuPDFCore.MuPDF.StandardOutputMessage [static]
```

This event is invoked when [RedirectOutput](#) has been called and the native [MuPDF](#) library writes to the standard output stream.

Definition at line 588 of file [MuPDF.cs](#).

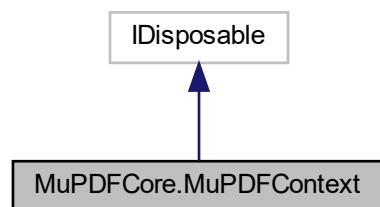
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDF.cs](#)

7.5 MuPDFCore.MuPDFContext Class Reference

A wrapper around a [MuPDF](#) context object, which contains the exception stack and the resource cache store.

Inheritance diagram for MuPDFCore.MuPDFContext:



Public Member Functions

- **MuPDFContext** (uint storeSize=256<< 20)
Create a new [MuPDFContext](#) instance with the specified cache store size.
- void **ClearStore** ()
Evict all items from the resource cache store (freeing the memory where they were held).
- void **ShrinkStore** (double fraction)
Evict items from the resource cache store (freeing the memory where they were held) until the the size of the store drops to the specified fraction of the current size.
- void **Dispose** ()

Properties

- long **StoreSize** [get]
The current size in bytes of the resource cache store. Read-only.
- long **StoreMaxSize** [get]
The maximum size in bytes of the resource cache store. Read-only.

7.5.1 Detailed Description

A wrapper around a [MuPDF](#) context object, which contains the exception stack and the resource cache store.

Definition at line 25 of file [MuPDFContext.cs](#).

7.5.2 Constructor & Destructor Documentation

7.5.2.1 MuPDFContext()

```
MuPDFCore.MuPDFContext.MuPDFContext (
    uint storeSize = 256 << 20 )
```

Create a new [MuPDFContext](#) instance with the specified cache store size.

Parameters

<code>storeSize</code>	The maximum size in bytes of the resource cache store. The default value is 256 MiB.
------------------------	--

Definition at line 58 of file [MuPDFContext.cs](#).

7.5.3 Member Function Documentation

7.5.3.1 ClearStore()

```
void MuPDFCore.MuPDFContext.ClearStore ( )
```

Evict all items from the resource cache store (freeing the memory where they were held).

Definition at line 87 of file [MuPDFContext.cs](#).

7.5.3.2 Dispose()

```
void MuPDFCore.MuPDFContext.Dispose ( )
```

Definition at line 132 of file [MuPDFContext.cs](#).

7.5.3.3 ShrinkStore()

```
void MuPDFCore.MuPDFContext.ShrinkStore ( 
    double fraction )
```

Evict items from the resource cache store (freeing the memory where they were held) until the size of the store drops to the specified fraction of the current size.

Parameters

<i>fraction</i>	The fraction of the current size that constitutes the target size of the store. If this is ≤ 0 , the cache is cleared. If this is ≥ 1 , nothing happens.
-----------------	--

Definition at line 96 of file [MuPDFContext.cs](#).

7.5.4 Property Documentation

7.5.4.1 StoreMaxSize

```
long MuPDFCore.MuPDFContext.StoreMaxSize [get]
```

The maximum size in bytes of the resource cache store. Read-only.

Definition at line 46 of file [MuPDFContext.cs](#).

7.5.4.2 StoreSize

```
long MuPDFCore.MuPDFContext.StoreSize [get]
```

The current size in bytes of the resource cache store. Read-only.

Definition at line 35 of file [MuPDFContext.cs](#).

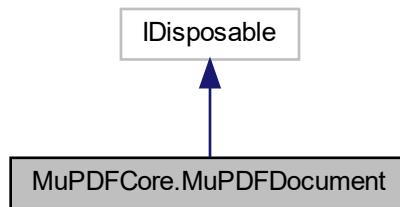
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFContext.cs](#)

7.6 MuPDFCore.MuPDFDocument Class Reference

A wrapper over a [MuPDF](#) document object, which contains possibly multiple pages.

Inheritance diagram for MuPDFCore.MuPDFDocument:



Public Member Functions

- **MuPDFDocument** ([MuPDFContext](#) context, IntPtr dataAddress, long dataLength, [InputFileTypes](#) fileType)

Create a new [MuPDFDocument](#) from data bytes accessible through the specified pointer.
- **MuPDFDocument** ([MuPDFContext](#) context, IntPtr dataAddress, long dataLength, [InputFileTypes](#) fileType, ref IDisposable dataHolder)

Create a new [MuPDFDocument](#) from data bytes accessible through the specified pointer.
- **MuPDFDocument** ([MuPDFContext](#) context, byte[] data, [InputFileTypes](#) fileType)

Create a new [MuPDFDocument](#) from an array of bytes.
- **MuPDFDocument** ([MuPDFContext](#) context, ref MemoryStream data, [InputFileTypes](#) fileType)

Create a new [MuPDFDocument](#) from a [MemoryStream](#).
- **MuPDFDocument** ([MuPDFContext](#) context, string fileName)

Create a new [MuPDFDocument](#) from a file.
- **void ClearCache ()**

Discard all the display lists that have been loaded from the document, possibly freeing some memory in the case of a huge document.
- **byte[] Render** (int pageNumber, [Rectangle](#) region, double zoom, [PixelFormats](#) pixelFormat, bool includeAnnotations=true)

Annotations=true)

- `byte[] Render (int pageNumber, double zoom, PixelFormats pixelFormat, bool includeAnnotations=true)`

Render (part of) a page to an array of bytes.
- `void Render (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, IntPtr destination, bool includeAnnotations=true)`

Render a page to an array of bytes.
- `void Render (int pageNumber, double zoom, PixelFormats pixelFormat, IntPtr destination, bool includeAnnotations=true)`

Render (part of) a page to the specified destination.
- `void Render (int pageNumber, double zoom, PixelFormats pixelFormat, IntPtr destination, bool includeAnnotations=true)`

Render a page to the specified destination.
- `Span< byte > Render (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, out IDisposable disposable, bool includeAnnotations=true)`

Render (part of) a page to a Span<byte>.
- `Span< byte > Render (int pageNumber, double zoom, PixelFormats pixelFormat, out IDisposable disposable, bool includeAnnotations=true)`

Render a page to a Span<byte>.
- `MuPDFMultiThreadedPageRenderer GetMultiThreadedRenderer (int pageNumber, int threadCount, bool includeAnnotations=true)`

Create a new MuPDFMultiThreadedPageRenderer that renders the specified page with the specified number of threads.
- `int GetRenderedSize (int pageNumber, double zoom, PixelFormats pixelFormat)`

Determine how many bytes will be necessary to render the specified page at the specified zoom level, using the the specified pixel format.
- `void SaveImage (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, string fileName, RasterOutputFileTypes fileType, bool includeAnnotations=true)`

Save (part of) a page to an image file in the specified format.
- `void SaveImage (int pageNumber, double zoom, PixelFormats pixelFormat, string fileName, RasterOutputFileTypes fileType, bool includeAnnotations=true)`

Save a page to an image file in the specified format.
- `void WriteImage (int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat, Stream outputStream, RasterOutputFileTypes fileType, bool includeAnnotations=true)`

Write (part of) a page to an image stream in the specified format.
- `void WriteImage (int pageNumber, double zoom, PixelFormats pixelFormat, Stream outputStream, RasterOutputFileTypes fileType, bool includeAnnotations=true)`

Write a page to an image stream in the specified format.
- `MuPDFStructuredTextPage GetStructuredTextPage (int pageNumber, bool includeAnnotations=true)`

Creates a new MuPDFStructuredTextPage from the specified page. This contains information about the text layout that can be used for highlighting and searching. The reading order is taken from the order the text is drawn in the source file, so may not be accurate.
- `MuPDFStructuredTextPage GetStructuredTextPage (int pageNumber, TesseractLanguage ocrLanguage, bool includeAnnotations=true, CancellationToken cancellationToken=default, IProgress< OCRProgressInfo > progress=null)`

Creates a new MuPDFStructuredTextPage from the specified page, using optical character recognition (OCR) to determine what text is written on the image. This contains information about the text layout that can be used for highlighting and searching.
- `async Task< MuPDFStructuredTextPage > GetStructuredTextPageAsync (int pageNumber, TesseractLanguage ocrLanguage, bool includeAnnotations=true, CancellationToken cancellationToken=default, IProgress< OCRProgressInfo > progress=null)`

Creates a new MuPDFStructuredTextPage from the specified page, using optical character recognition (OCR) to determine what text is written on the image. This contains information about the text layout that can be used for highlighting and searching. The OCR step is run asynchronously, e.g. to avoid blocking the UI thread.
- `string ExtractText (string separator=null, bool includeAnnotations=true)`

Extracts all the text from the document and returns it as a string. The reading order is taken from the order the text is drawn in the source file, so may not be accurate.
- `string ExtractText (TesseractLanguage ocrLanguage, string separator=null, bool includeAnnotations=true)`

- *Extracts all the text from the document and returns it as a string, using optical character recognition (OCR) to determine what text is written on the image.*
- `async Task< string > ExtractTextAsync (TesseractLanguage ocrLanguage, string separator=null, bool includeAnnotations=true, CancellationToken cancellationToken=default, IProgress< OCRProgressInfo > progress=null)`
Extracts all the text from the document and returns it as a string, using optical character recognition (OCR) to determine what text is written on the image. The OCR step is run asynchronously, e.g. to avoid blocking the UI thread.
- `bool TryUnlock (string password)`
Attempts to unlock the document with the supplied password.
- `bool TryUnlock (string password, out PasswordTypes passwordType)`
Attempts to unlock the document with the supplied password.
- `void Dispose ()`

Static Public Member Functions

- `static int GetRenderedSize (Rectangle region, double zoom, PixelFormats pixelFormat)`
Determine how many bytes will be necessary to render the specified region in page units at the specified zoom level, using the the specified pixel format.
- `static void CreateDocument (MuPDFContext context, string fileName, DocumentOutputFileTypes fileType, bool includeAnnotations=true, params(MuPDFPage page, Rectangle region, float zoom)[] pages)`
Create a new document containing the specified (parts of) pages from other documents.
- `static void CreateDocument (MuPDFContext context, string fileName, DocumentOutputFileTypes fileType, bool includeAnnotations=true, params MuPDFPage[] pages)`
Create a new document containing the specified pages from other documents.

Properties

- `MuPDFPageCollection Pages [get]`
The pages contained in the document.
- `bool ClipToPageBounds = true [get, set]`
Defines whether the images resulting from rendering operations should be clipped to the page boundaries.
- `EncryptionState EncryptionState [get]`
Describes the encryption state of the document.
- `RestrictionState RestrictionState [get]`
Describes the restriction state of the document.
- `DocumentRestrictions Restrictions [get]`
Describes the operations that are restricted on the document. This is not actually enforced by the library, but library users should only allow these operations if the document has been unlocked with the owner password (i.e. if `RestrictionState` is `RestrictionState.Unlocked`).

7.6.1 Detailed Description

A wrapper over a `MuPDF` document object, which contains possibly multiple pages.

Definition at line 30 of file `MuPDFDocument.cs`.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 MuPDFDocument() [1/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
    MuPDFContext context,
    IntPtr dataAddress,
    long dataLength,
    InputFileTypes fileType )
```

Create a new [MuPDFDocument](#) from data bytes accessible through the specified pointer.

Parameters

<i>context</i>	The context that will own this document.
<i>dataAddress</i>	A pointer to the data bytes that make up the document.
<i>dataLength</i>	The number of bytes to read from the specified address.
<i>fileType</i>	The type of the document to read.

Definition at line 135 of file [MuPDFDocument.cs](#).

7.6.2.2 MuPDFDocument() [2/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
    MuPDFContext context,
    IntPtr dataAddress,
    long dataLength,
    InputFileTypes fileType,
    ref IDisposable dataHolder )
```

Create a new [MuPDFDocument](#) from data bytes accessible through the specified pointer.

Parameters

<i>context</i>	The context that will own this document.
<i>dataAddress</i>	A pointer to the data bytes that make up the document.
<i>dataLength</i>	The number of bytes to read from the specified address.
<i>fileType</i>	The type of the document to read.
<i>dataHolder</i>	An IDisposable that will be disposed when the MuPDFDocument is disposed.

Definition at line 145 of file [MuPDFDocument.cs](#).

7.6.2.3 MuPDFDocument() [3/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
    MuPDFContext context,
    byte[] data,
    InputFileTypes fileType )
```

Create a new [MuPDFDocument](#) from an array of bytes.

Parameters

<i>context</i>	The context that will own this document.
<i>data</i>	An array containing the data bytes that make up the document. This must not be altered until after the MuPDFDocument has been disposed! The address of the array will be pinned, which may cause degradation in the Garbage Collector's performance, and is thus only advised for short-lived documents. To avoid this issue, marshal the bytes to unmanaged memory and use one of the IntPtr constructors.
<i>fileType</i>	The type of the document to read.

Definition at line 245 of file [MuPDFDocument.cs](#).

7.6.2.4 MuPDFDocument() [4/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
    MuPDFContext context,
    ref MemoryStream data,
    InputFileTypes fileType )
```

Create a new [MuPDFDocument](#) from a [MemoryStream](#).

Parameters

<i>context</i>	The context that will own this document.
<i>data</i>	The MemoryStream containing the data that makes up the document. This will be disposed when the MuPDFDocument has been disposed and must not be disposed externally! The address of the MemoryStream 's buffer will be pinned, which may cause degradation in the Garbage Collector's performance, and is thus only advised for short-lived documents. To avoid this issue, marshal the bytes to unmanaged memory and use one of the IntPtr constructors.
<i>fileType</i>	The type of the document to read.

Definition at line 347 of file [MuPDFDocument.cs](#).

7.6.2.5 MuPDFDocument() [5/5]

```
MuPDFCore.MuPDFDocument.MuPDFDocument (
    MuPDFContext context,
    string fileName )
```

Create a new [MuPDFDocument](#) from a file.

Parameters

<i>context</i>	The context that will own this document.
<i>fileName</i>	The path to the file to open.

Definition at line 453 of file [MuPDFDocument.cs](#).

7.6.3 Member Function Documentation

7.6.3.1 ClearCache()

```
void MuPDFCore.MuPDFDocument.ClearCache ( )
```

Discard all the display lists that have been loaded from the document, possibly freeing some memory in the case of a huge document.

Definition at line 584 of file [MuPDFDocument.cs](#).

7.6.3.2 CreateDocument() [1/2]

```
static void MuPDFCore.MuPDFDocument.CreateDocument (
    MuPDFContext context,
    string fileName,
    DocumentOutputFileTypes fileType,
    bool includeAnnotations = true,
    params MuPDFPage[] pages ) [static]
```

Create a new document containing the specified pages from other documents.

Parameters

<code>context</code>	The context that was used to open the documents.
<code>fileName</code>	The output file name.
<code>fileType</code>	The output file format.
<code>pages</code>	The pages to include in the document.
<code>includeAnnotations</code>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1135 of file [MuPDFDocument.cs](#).

7.6.3.3 CreateDocument() [2/2]

```
static void MuPDFCore.MuPDFDocument.CreateDocument (
    MuPDFContext context,
    string fileName,
    DocumentOutputFileTypes fileType,
```

```
bool includeAnnotations = true,  
params (MuPDFPage page, Rectangle region, float zoom) [ ] pages ) [static]
```

Create a new document containing the specified (parts of) pages from other documents.

Parameters

<i>context</i>	The context that was used to open the documents.
<i>fileName</i>	The output file name.
<i>fileType</i>	The output file format.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.
<i>pages</i>	The pages to include in the document. The "page" element specifies the page, the "region" element the area of the page that should be included in the document, and the "zoom" element how much the region should be scaled.

Definition at line 1027 of file [MuPDFDocument.cs](#).

7.6.3.4 Dispose()

```
void MuPDFCore.MuPDFDocument.Dispose ( )
```

Definition at line 1494 of file [MuPDFDocument.cs](#).

7.6.3.5 ExtractText() [1/2]

```
string MuPDFCore.MuPDFDocument.ExtractText (
    string separator = null,
    bool includeAnnotations = true )
```

Extracts all the text from the document and returns it as a string. The reading order is taken from the order the text is drawn in the source file, so may not be accurate.

Parameters

<i>separator</i>	The character(s) used to separate the text lines obtained from the document. If this is <code>null</code> , <code>Environment.NewLine</code> is used as a default separator.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included. Otherwise, only the page contents are included.

Returns

A string containing all the text in the document. Characters are converted from the UTF-8 representation used in the document to equivalent UTF-16 strings.

Definition at line 1241 of file [MuPDFDocument.cs](#).

7.6.3.6 ExtractText() [2/2]

```
string MuPDFCore.MuPDFDocument.ExtractText (
    TesseractLanguage ocrLanguage,
    string separator = null,
    bool includeAnnotations = true )
```

Extracts all the text from the document and returns it as a string, using optical character recognition (OCR) to determine what text is written on the image.

Parameters

<i>separator</i>	The character(s) used to separate the text lines obtained from the document. If this is null, Environment.NewLine is used as a default separator.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
<i>includeAnnotations</i>	If this is true, annotations (e.g. signatures) are included. Otherwise, only the page contents are included.

Returns

A string containing all the text in the document. Characters are converted from the UTF-8 representation used in the document to equivalent UTF-16 strings.

Definition at line 1288 of file [MuPDFDocument.cs](#).

7.6.3.7 ExtractTextAsync()

```
async Task< string > MuPDFCore.MuPDFDocument.ExtractTextAsync (
    TesseractLanguage ocrLanguage,
    string separator = null,
    bool includeAnnotations = true,
    CancellationToken cancellationToken = default,
    IProgress< OCRProgressInfo > progress = null )
```

Extracts all the text from the document and returns it as a string, using optical character recognition (OCR) to determine what text is written on the image. The OCR step is run asynchronously, e.g. to avoid blocking the UI thread.

Parameters

<i>separator</i>	The character(s) used to separate the text lines obtained from the document. If this is null, Environment.NewLine is used as a default separator.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
<i>includeAnnotations</i>	If this is true, annotations (e.g. signatures) are included. Otherwise, only the page contents are included.
<i>cancellationToken</i>	A CancellationToken used to cancel the operation.
<i>progress</i>	An IProgress<OCRProgressInfo> used to report progress.

Returns

A string containing all the text in the document. Characters are converted from the UTF-8 representation used in the document to equivalent UTF-16 strings.

Definition at line 1337 of file [MuPDFDocument.cs](#).

7.6.3.8 GetMultiThreadedRenderer()

```
MuPDFMultiThreadedPageRenderer MuPDFCore.MuPDFDocument.GetMultiThreadedRenderer (
    int pageNumber,
    int threadCount,
    bool includeAnnotations = true )
```

Create a new [MuPDFMultiThreadedPageRenderer](#) that renders the specified page with the specified number of threads.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>threadCount</i>	The number of threads to use. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than <i>threadCount</i> that satisfies this condition is used.

Returns

A [MuPDFMultiThreadedPageRenderer](#) that can be used to render the specified page with the specified number of threads.

Parameters

<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.
---------------------------	--

Definition at line 782 of file [MuPDFDocument.cs](#).

7.6.3.9 GetRenderedSize() [1/2]

```
int MuPDFCore.MuPDFDocument.GetRenderedSize (
    int pageNumber,
    double zoom,
    PixelFormats pixelFormat )
```

Determine how many bytes will be necessary to render the specified page at the specified zoom level, using the the specified pixel format.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixels data.

Returns

An integer representing the number of bytes that will be necessary to store the pixel data of the rendered image.

Definition at line 804 of file [MuPDFDocument.cs](#).

7.6.3.10 GetRenderedSize() [2/2]

```
static int MuPDFCore.MuPDFDocument.GetRenderedSize (
    Rectangle region,
    double zoom,
    PixelFormats pixelFormat ) [static]
```

Determine how many bytes will be necessary to render the specified region in page units at the specified zoom level, using the the specified pixel format.

Parameters

<i>region</i>	The region that will be rendered.
<i>zoom</i>	The scale at which the region will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixels data.

Returns

An integer representing the number of bytes that will be necessary to store the pixel data of the rendered image.

Definition at line 821 of file [MuPDFDocument.cs](#).

7.6.3.11 GetStructuredTextPage() [1/2]

```
MuPDFStructuredTextPage MuPDFCore.MuPDFDocument.GetStructuredTextPage (
    int pageNumber,
    bool includeAnnotations = true )
```

Creates a new [MuPDFStructuredTextPage](#) from the specified page. This contains information about the text layout that can be used for highlighting and searching. The reading order is taken from the order the text is drawn in the source file, so may not be accurate.

Parameters

<code>pageNumber</code>	The number of the page (starting at 0)
<code>includeAnnotations</code>	If this is <code>true</code> , annotations (e.g. signatures) are included. Otherwise, only the page contents are included.

Returns

A [MuPDFStructuredTextPage](#) containing a structured text representation of the page.

Definition at line 1154 of file [MuPDFDocument.cs](#).

7.6.3.12 GetStructuredTextPage() [2/2]

```
MuPDFStructuredTextPage MuPDFCore.MuPDFDocument.GetStructuredTextPage (
    int pageNumber,
    TesseractLanguage ocrLanguage,
    bool includeAnnotations = true,
    CancellationToken cancellationToken = default,
    IProgress<OCRProgressInfo> progress = null )
```

Creates a new [MuPDFStructuredTextPage](#) from the specified page, using optical character recognition (OCR) to determine what text is written on the image. This contains information about the text layout that can be used for highlighting and searching.

Parameters

<code>pageNumber</code>	The number of the page (starting at 0)
<code>ocrLanguage</code>	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
<code>includeAnnotations</code>	If this is <code>true</code> , annotations (e.g. signatures) are included. Otherwise, only the page contents are included.
<code>cancellationToken</code>	A <code>CancellationToken</code> used to cancel the operation. Providing a value other than the default is not supported on Windows x86 and will throw a runtime exception.
<code>progress</code>	An <code>IProgress<OCRProgressInfo></code> used to report progress. Providing a value other than null is not supported on Windows x86 and will throw a runtime exception.

Returns

A [MuPDFStructuredTextPage](#) containing a structured text representation of the page.

Definition at line 1178 of file [MuPDFDocument.cs](#).

7.6.3.13 GetStructuredTextPageAsync()

```
async Task<MuPDFStructuredTextPage> MuPDFCore.MuPDFDocument.GetStructuredTextPageAsync (
    int pageNumber,
```

```
TesseractLanguage ocrLanguage,
bool includeAnnotations = true,
CancellationToken cancellationToken = default,
IProgress< OCRProgressInfo > progress = null )
```

Creates a new [MuPDFStructuredTextPage](#) from the specified page, using optical character recognition (OCR) to determine what text is written on the image. This contains information about the text layout that can be used for highlighting and searching. The OCR step is run asynchronously, e.g. to avoid blocking the UI thread.

Parameters

<i>pageNumber</i>	The number of the page (starting at 0)
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included. Otherwise, only the page contents are included.
<i>cancellationToken</i>	A <code>CancellationToken</code> used to cancel the operation. Providing a value other than the default is not supported on Windows x86 and will throw a runtime exception.
<i>progress</i>	An <code>IProgress<OCRProgressInfo></code> used to report progress. Providing a value other than null is not supported on Windows x86 and will throw a runtime exception.

Returns

A [MuPDFStructuredTextPage](#) containing a structured text representation of the page.

Definition at line 1211 of file [MuPDFDocument.cs](#).

7.6.3.14 Render() [1/6]

```
byte[ ] MuPDFCore.MuPDFDocument.Render (
    int pageNumber,
    double zoom,
    PixelFormats pixelFormat,
    bool includeAnnotations = true )
```

Render a page to an array of bytes.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Returns

A byte array containing the raw values for the pixels of the rendered image.

Definition at line 636 of file [MuPDFDocument.cs](#).

7.6.3.15 Render() [2/6]

```
void MuPDFCore.MuPDFDocument.Render (
    int pageNumber,
    double zoom,
    PixelFormats pixelFormat,
    IntPtr destination,
    bool includeAnnotations = true )
```

Render a page to the specified destination.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>destination</i>	The address of the buffer where the pixel data will be written. There must be enough space available to write the values for all the pixels, otherwise this will fail catastrophically!
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 715 of file [MuPDFDocument.cs](#).

7.6.3.16 Render() [3/6]

```
Span< byte > MuPDFCore.MuPDFDocument.Render (
    int pageNumber,
    double zoom,
    PixelFormats pixelFormat,
    out IDisposable disposable,
    bool includeAnnotations = true )
```

Render a page to a `Span<byte>`.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>disposable</i>	An <code>IDisposable</code> that can be used to free the memory where the image is stored. You should keep track of this and dispose it when you have finished working with the image.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 763 of file [MuPDFDocument.cs](#).

7.6.3.17 Render() [4/6]

```
byte[] MuPDFCore.MuPDFDocument.Render (
    int pageNumber,
    Rectangle region,
    double zoom,
    PixelFormats pixelFormat,
    bool includeAnnotations = true )
```

Render (part of) a page to an array of bytes.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>region</i>	The region of the page to render in page units.
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Returns

A byte array containing the raw values for the pixels of the rendered image.

Definition at line 602 of file [MuPDFDocument.cs](#).

7.6.3.18 Render() [5/6]

```
void MuPDFCore.MuPDFDocument.Render (
    int pageNumber,
    Rectangle region,
    double zoom,
    PixelFormats pixelFormat,
    IntPtr destination,
    bool includeAnnotations = true )
```

Render (part of) a page to the specified destination.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>region</i>	The region of the page to render in page units.
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>destination</i>	The address of the buffer where the pixel data will be written. There must be enough space available to write the values for all the pixels, otherwise this will fail catastrophically!
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 656 of file [MuPDFDocument.cs](#).

7.6.3.19 Render() [6/6]

```
Span< byte > MuPDFCore.MuPDFDocument.Render (
    int pageNumber,
    Rectangle region,
    double zoom,
    PixelFormats pixelFormat,
    out IDisposable disposable,
    bool includeAnnotations = true )
```

Render (part of) a page to a Span<byte>.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>region</i>	The region of the page to render in page units.
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>disposable</i>	An IDisposable that can be used to free the memory where the image is stored. You should keep track of this and dispose it when you have finished working with the image.
<i>includeAnnotations</i>	If this is true, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 735 of file [MuPDFDocument.cs](#).

7.6.3.20 SaveImage() [1/2]

```
void MuPDFCore.MuPDFDocument.SaveImage (
    int pageNumber,
    double zoom,
    PixelFormats pixelFormat,
    string fileName,
    RasterOutputFileTypes fileType,
    bool includeAnnotations = true )
```

Save a page to an image file in the specified format.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>fileName</i>	The path to the output file.
<i>fileType</i>	The output format of the file.
<i>includeAnnotations</i>	If this is true, annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 916 of file [MuPDFDocument.cs](#).

7.6.3.21 SaveImage() [2/2]

```
void MuPDFCore.MuPDFDocument.SaveImage (
    int pageNumber,
    Rectangle region,
    double zoom,
    PixelFormats pixelFormat,
    string fileName,
    RasterOutputFileTypes fileType,
    bool includeAnnotations = true )
```

Save (part of) a page to an image file in the specified format.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>region</i>	The region of the page to render in page units.
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>fileName</i>	The path to the output file.
<i>fileType</i>	The output format of the file.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 857 of file [MuPDFDocument.cs](#).

7.6.3.22 TryUnlock() [1/2]

```
bool MuPDFCore.MuPDFDocument.TryUnlock (
    string password )
```

Attempts to unlock the document with the supplied password.

Parameters

<i>password</i>	The user or owner password to use to unlock the document.
-----------------	---

Returns

`true` if the document was successfully unlocked (or if it was never locked to begin with), `false` if the password was incorrect and the document is still locked.

This method can be used both to unlock an encrypted document and to check whether the supplied owner password is correct.

Definition at line 1383 of file [MuPDFDocument.cs](#).

7.6.3.23 TryUnlock() [2/2]

```
bool MuPDFCore.MuPDFDocument.TryUnlock (
    string password,
    out PasswordTypes passwordType )
```

Attempts to unlock the document with the supplied password.

Parameters

<i>password</i>	The user or owner password to use to unlock the document.
<i>passwordType</i>	If the method returns <code>true</code> , this can be used to determine whether the supplied password was the user password or the owner password. If the method returns <code>false</code> , this can be used to determine whether a user password and/or an owner password are required.

Returns

`true` if the document was successfully unlocked (or if it was never locked to begin with), `false` if the password was incorrect and the document is still locked.

This method can be used both to unlock an encrypted document and to check whether the supplied owner password is correct.

Definition at line 1396 of file [MuPDFDocument.cs](#).

7.6.3.24 WriteImage() [1/2]

```
void MuPDFCore.MuPDFDocument.WriteImage (
    int pageNumber,
    double zoom,
    PixelFormats pixelFormat,
    Stream outputStream,
    RasterOutputFileTypes fileType,
    bool includeAnnotations = true )
```

Write a page to an image stream in the specified format.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>outputStream</i>	The stream to which the image data will be written.
<i>fileType</i>	The output format of the image.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 1008 of file [MuPDFDocument.cs](#).

7.6.3.25 WriteImage() [2/2]

```
void MuPDFCore.MuPDFDocument.WriteImage (
    int pageNumber,
    Rectangle region,
    double zoom,
    PixelFormats pixelFormat,
    Stream outputStream,
    RasterOutputFileTypes fileType,
    bool includeAnnotations = true )
```

Write (part of) a page to an image stream in the specified format.

Parameters

<i>pageNumber</i>	The number of the page to render (starting at 0).
<i>region</i>	The region of the page to render in page units.
<i>zoom</i>	The scale at which the page will be rendered. This will determine the size in pixel of the image.
<i>pixelFormat</i>	The format of the pixel data.
<i>outputStream</i>	The stream to which the image data will be written.
<i>fileType</i>	The output format of the image.
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the display list that is generated. Otherwise, only the page contents are included.

Definition at line 937 of file [MuPDFDocument.cs](#).

7.6.4 Property Documentation

7.6.4.1 ClipToBounds

```
bool MuPDFCore.MuPDFDocument.ClipToBounds = true [get], [set]
```

Defines whether the images resulting from rendering operations should be clipped to the page boundaries.

Definition at line 109 of file [MuPDFDocument.cs](#).

7.6.4.2 EncryptionState

```
EncryptionState MuPDFCore.MuPDFDocument.EncryptionState [get]
```

Describes the encryption state of the document.

Definition at line 114 of file [MuPDFDocument.cs](#).

7.6.4.3 Pages

```
MuPDFPageCollection MuPDFCore.MuPDFDocument.Pages [get]
```

The pages contained in the document.

Definition at line 104 of file [MuPDFDocument.cs](#).

7.6.4.4 Restrictions

```
DocumentRestrictions MuPDFCore.MuPDFDocument.Restrictions [get]
```

Describes the operations that are restricted on the document. This is not actually enforced by the library, but library users should only allow these operations if the document has been unlocked with the owner password (i.e. if [RestrictionState](#) is [RestrictionState.Unlocked](#)).

Definition at line 126 of file [MuPDFDocument.cs](#).

7.6.4.5 RestrictionState

```
RestrictionState MuPDFCore.MuPDFDocument.RestrictionState [get]
```

Describes the restriction state of the document.

Definition at line 119 of file [MuPDFDocument.cs](#).

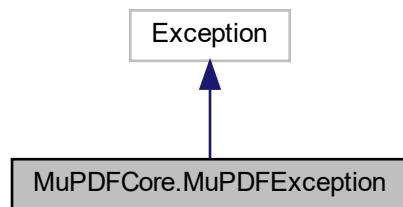
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFDocument.cs](#)

7.7 MuPDFCore.MuPDFException Class Reference

The exception that is thrown when a [MuPDF](#) operation fails.

Inheritance diagram for MuPDFCore.MuPDFException:



Public Attributes

- readonly [ExitCodes ErrorCode](#)

The [ExitCodes](#) returned by the native function.

7.7.1 Detailed Description

The exception that is thrown when a [MuPDF](#) operation fails.

Definition at line [478](#) of file [MuPDF.cs](#).

7.7.2 Member Data Documentation

7.7.2.1 ErrorCode

```
readonly ExitCodes MuPDFCore.MuPDFException.ErrorCode
```

The [ExitCodes](#) returned by the native function.

Definition at line [483](#) of file [MuPDF.cs](#).

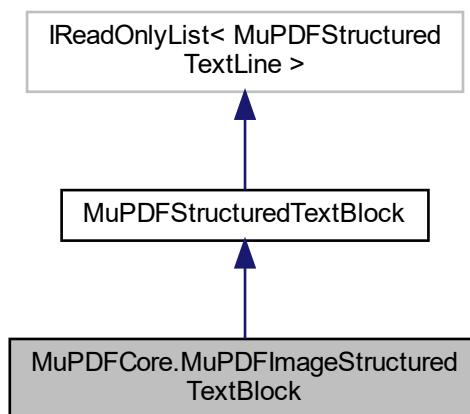
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDF.cs](#)

7.8 MuPDFCore.MuPDFImageStructuredTextBlock Class Reference

Represents a block containing a single image. The block contains a single line with a single character.

Inheritance diagram for MuPDFCore.MuPDFImageStructuredTextBlock:



Public Member Functions

- override `IEnumerator< MuPDFStructuredTextLine > GetEnumerator ()`

Properties

- override `Types Type [get]`
- override `int Count [get]`
- override `MuPDFStructuredTextLine this[int index] [get]`

Additional Inherited Members

7.8.1 Detailed Description

Represents a block containing a single image. The block contains a single line with a single character.

Definition at line [615](#) of file [MuPDFStructuredTextPage.cs](#).

7.8.2 Member Function Documentation

7.8.2.1 GetEnumerator()

```
override IEnumerator< MuPDFStructuredTextLine > MuPDFCore.MuPDFImageStructuredTextBlock.GetEnumerator ( ) [virtual]
```

Implements [MuPDFCore.MuPDFStructuredTextBlock](#).

Definition at line [647](#) of file [MuPDFStructuredTextPage.cs](#).

7.8.3 Property Documentation

7.8.3.1 Count

```
override int MuPDFCore.MuPDFImageStructuredTextBlock.Count [get]
```

Definition at line [621](#) of file [MuPDFStructuredTextPage.cs](#).

7.8.3.2 this[int index]

```
override MuPDFStructuredTextLine MuPDFCore.MuPDFImageStructuredTextBlock.this[int index] [get]
Definition at line 626 of file MuPDFStructuredTextPage.cs.
```

7.8.3.3 Type

```
override Types MuPDFCore.MuPDFImageStructuredTextBlock.Type [get]
```

Definition at line 618 of file MuPDFStructuredTextPage.cs.

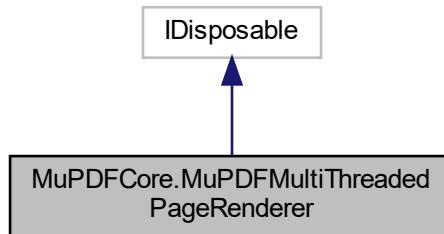
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.9 MuPDFCore.MuPDFMultiThreadedPageRenderer Class Reference

A class that holds the necessary resources to render a page of a [MuPDF](#) document using multiple threads.

Inheritance diagram for MuPDFCore.MuPDFMultiThreadedPageRenderer:



Public Member Functions

- void [Render \(RoundedSize targetSize, Rectangle region, IntPtr\[\] destinations, PixelFormats pixelFormat\)](#)
Render the specified region to an image of the specified size, split in a number of tiles equal to the number of threads used by this [MuPDFMultiThreadedPageRenderer](#), without marshaling. This method will not return until all the rendering threads have finished.
- delegate Span< byte > [GetSpanItem \(int index\)](#)
Gets an element from a collection of [Span<byte>](#)
- [GetSpanItem Render \(RoundedSize targetSize, Rectangle region, out IDisposable\[\] disposables, PixelFormats pixelFormat\)](#)
Render the specified region to an image of the specified size, split in a number of tiles equal to the number of threads used by this [MuPDFMultiThreadedPageRenderer](#), without marshaling. This method will not return until all the rendering threads have finished. Since creating an array of [Span<T>](#) is not allowed, this method returns a delegate that accepts an integer parameter (representing the index of the span in the "array") and returns the [Span<T>](#) corresponding to that index.
- void [Abort \(\)](#)
Signal to the rendering threads that they should abort rendering as soon as possible.
- [RenderProgress GetProgress \(\)](#)
Get the current rendering progress of all the threads.
- void [Dispose \(\)](#)

Properties

- int [ThreadCount](#) [get]
The number of threads that are used to render the image.

7.9.1 Detailed Description

A class that holds the necessary resources to render a page of a MuPDF document using multiple threads.

Definition at line 276 of file [MuPDFMultiThreadedPageRenderer.cs](#).

7.9.2 Member Function Documentation

7.9.2.1 Abort()

```
void MuPDFCore.MuPDFMultiThreadedPageRenderer.Abort ( )
```

Signal to the rendering threads that they should abort rendering as soon as possible.

Definition at line 543 of file [MuPDFMultiThreadedPageRenderer.cs](#).

7.9.2.2 Dispose()

```
void MuPDFCore.MuPDFMultiThreadedPageRenderer.Dispose ( )
```

Definition at line 592 of file [MuPDFMultiThreadedPageRenderer.cs](#).

7.9.2.3 GetProgress()

```
RenderProgress MuPDFCore.MuPDFMultiThreadedPageRenderer.GetProgress ( )
```

Get the current rendering progress of all the threads.

Returns

A [RenderProgress](#) object containing the rendering progress of all the threads.

Definition at line 555 of file [MuPDFMultiThreadedPageRenderer.cs](#).

7.9.2.4 GetSpanItem()

```
delegate Span< byte > MuPDFCore.MuPDFMultiThreadedPageRenderer.GetSpanItem ( int index )
```

Gets an element from a collection of `Span<byte>`

Parameters

<i>index</i>	The index of the element to get.
--------------	----------------------------------

Returns

An element from a collection of `Span<byte>`

7.9.2.5 Render() [1/2]

```
void MuPDFCore.MuPDFMultiThreadedPageRenderer.Render (
    RoundedSize targetSize,
    Rectangle region,
    IntPtr[] destinations,
    PixelFormats pixelFormat )
```

Render the specified region to an image of the specified size, split in a number of tiles equal to the number of threads used by this [MuPDFMultiThreadedPageRenderer](#), without marshaling. This method will not return until all the rendering threads have finished.

Parameters

<i>targetSize</i>	The total size of the image that should be rendered.
<i>region</i>	The region in page units that should be rendered.
<i>destinations</i>	An array containing the addresses of the buffers where the rendered tiles will be written. There must be enough space available in each buffer to write the values for all the pixels of the tile, otherwise this will fail catastrophically! As long as the <i>targetSize</i> is the same, the size in pixel of the tiles is guaranteed to also be the same.
<i>pixelFormat</i>	The format of the pixel data.

Definition at line 383 of file [MuPDFMultiThreadedPageRenderer.cs](#).

7.9.2.6 Render() [2/2]

```
GetSpanItem MuPDFCore.MuPDFMultiThreadedPageRenderer.Render (
    RoundedSize targetSize,
    Rectangle region,
    out IDisposable[] disposables,
    PixelFormats pixelFormat )
```

Render the specified region to an image of the specified size, split in a number of tiles equal to the number of threads used by this [MuPDFMultiThreadedPageRenderer](#), without marshaling. This method will not return until all the rendering threads have finished. Since creating an array of `Span<T>` is not allowed, this method returns a delegate that accepts an integer parameter (representing the index of the span in the "array") and returns the `Span<T>` corresponding to that index.

Parameters

<i>targetSize</i>	The total size of the image that should be rendered.
<i>region</i>	The region in page units that should be rendered.
<i>disposables</i>	A collection of <code>IDisposable</code> s that can be used to free the memory where the rendered tiles are stored. You should keep track of these and dispose them when you have finished working with the images.
<i>pixelFormat</i>	The format of the pixel data.

Returns

A delegate that accepts an integer parameter (representing the index of the span in the "array") and returns the `Span<T>` corresponding to that index.

Definition at line 509 of file [MuPDFMultiThreadedPageRenderer.cs](#).

7.9.3 Property Documentation

7.9.3.1 ThreadCount

```
int MuPDFCore.MuPDFMultiThreadedPageRenderer.ThreadCount [get]
```

The number of threads that are used to render the image.

Definition at line 306 of file [MuPDFMultiThreadedPageRenderer.cs](#).

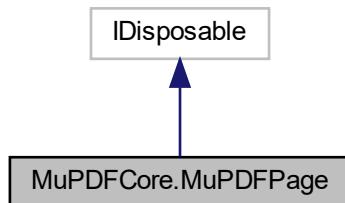
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFMultiThreadedPageRenderer.cs](#)

7.10 MuPDFCore.MuPDFPage Class Reference

A wrapper over a [MuPDF](#) page object, which contains information about the page's boundaries.

Inheritance diagram for `MuPDFCore.MuPDFPage`:



Public Member Functions

- void [Dispose \(\)](#)

Properties

- [Rectangle Bounds \[get\]](#)
The page's bounds at 72 DPI. Read-only.
- int [PageNumber \[get\]](#)
The number of this page in the original document.

7.10.1 Detailed Description

A wrapper over a [MuPDF](#) page object, which contains information about the page's boundaries.

Definition at line [27](#) of file [MuPDFPage.cs](#).

7.10.2 Member Function Documentation

7.10.2.1 Dispose()

```
void MuPDFCore.MuPDFPage.Dispose ( )
```

Definition at line [121](#) of file [MuPDFPage.cs](#).

7.10.3 Property Documentation

7.10.3.1 Bounds

[Rectangle](#) MuPDFCore.MuPDFPage.Bounds [get]

The page's bounds at 72 DPI. Read-only.

Definition at line [32](#) of file [MuPDFPage.cs](#).

7.10.3.2 PageNumber

```
int MuPDFCore.MuPDFPage.PageNumber [get]
```

The number of this page in the original document.

Definition at line 37 of file [MuPDFPage.cs](#).

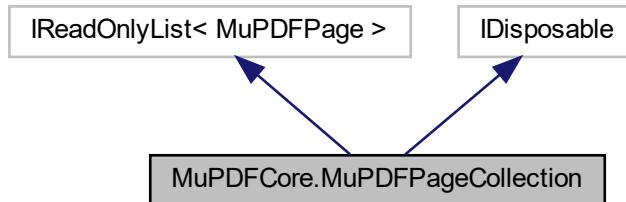
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFPage.cs](#)

7.11 MuPDFCore.MuPDFPageCollection Class Reference

A lazy collection of [MuPDFPages](#). Each page is loaded from the document as it is requested for the first time.

Inheritance diagram for MuPDFCore.MuPDFPageCollection:



Public Member Functions

- `IEnumerator< MuPDFPage > GetEnumerator ()`
inheritDoc/
- `void Dispose ()`

Properties

- `int Length [get]`
The number of pages in the collection.
- `int Count [get]`
The number of pages in the collection.
- `MuPDFPage this[int index] [get]`
Get a page from the collection.

7.11.1 Detailed Description

A lazy collection of [MuPDFPage](#)s. Each page is loaded from the document as it is requested for the first time.

Definition at line 131 of file [MuPDFPage.cs](#).

7.11.2 Member Function Documentation

7.11.2.1 Dispose()

```
void MuPDFCore.MuPDFPageCollection.Dispose ( )
```

Definition at line 234 of file [MuPDFPage.cs](#).

7.11.2.2 GetEnumerator()

```
IEnumerator< MuPDFPage > MuPDFCore.MuPDFPageCollection.GetEnumerator ( )  
inheritdoc/
```

Definition at line 195 of file [MuPDFPage.cs](#).

7.11.3 Property Documentation

7.11.3.1 Count

```
int MuPDFCore.MuPDFPageCollection.Count [get]
```

The number of pages in the collection.

Definition at line 156 of file [MuPDFPage.cs](#).

7.11.3.2 Length

```
int MuPDFCore.MuPDFPageCollection.Length [get]
```

The number of pages in the collection.

Definition at line 151 of file [MuPDFPage.cs](#).

7.11.3.3 this[int index]

```
MuPDFPage MuPDFCore.MuPDFPageCollection.this[int index] [get]
```

Get a page from the collection.

Parameters

<i>index</i>	The number of the page (starting at 0).
--------------	---

Returns

The specified [MuPDFPage](#).

Definition at line 163 of file [MuPDFPage.cs](#).

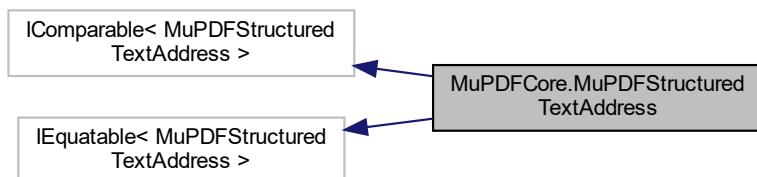
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFPage.cs

7.12 MuPDFCore.MuPDFStructuredTextAddress Struct Reference

Represents the address of a particular character in a [MuPDFStructuredTextPage](#), in terms of block index, line index and character index.

Inheritance diagram for MuPDFCore.MuPDFStructuredTextAddress:



Public Member Functions

- [MuPDFStructuredTextAddress](#) (int blockIndex, int lineIndex, int characterIndex)
Creates a new [MuPDFStructuredTextAddress](#) from the specified indices.
- int [CompareTo](#) ([MuPDFStructuredTextAddress](#) other)
Compares this [MuPDFStructuredTextAddress](#) with another [MuPDFStructuredTextAddress](#).
- override int [GetHashCode](#) ()
- [MuPDFStructuredTextAddress?](#) [Increment](#) ([MuPDFStructuredTextPage](#) page)
Returns a [MuPDFStructuredTextAddress](#) corresponding to the next character in the specified page.
- bool [Equals](#) ([MuPDFStructuredTextAddress](#) other)
Compares the current [MuPDFStructuredTextAddress](#) with another [MuPDFStructuredTextAddress](#).
- override bool [Equals](#) (object other)

Static Public Member Functions

- static bool `operator>` (`MuPDFStructuredTextAddress` first, `MuPDFStructuredTextAddress` second)
Compares two `MuPDFStructuredTextAddress`.
- static bool `operator>=` (`MuPDFStructuredTextAddress` first, `MuPDFStructuredTextAddress` second)
Compares two `MuPDFStructuredTextAddress`.
- static bool `operator<` (`MuPDFStructuredTextAddress` first, `MuPDFStructuredTextAddress` second)
Compares two `MuPDFStructuredTextAddress`.
- static bool `operator<=` (`MuPDFStructuredTextAddress` first, `MuPDFStructuredTextAddress` second)
Compares two `MuPDFStructuredTextAddress`.
- static bool `operator==` (`MuPDFStructuredTextAddress` first, `MuPDFStructuredTextAddress` second)
Compares two `MuPDFStructuredTextAddress`.
- static bool `operator!=` (`MuPDFStructuredTextAddress` first, `MuPDFStructuredTextAddress` second)
Compares two `MuPDFStructuredTextAddress`.

Public Attributes

- readonly int `BlockIndex`
The index of the block.
- readonly int `LineIndex`
The index of the line within the block.
- readonly int `CharacterIndex`
The index of the character within the line.

7.12.1 Detailed Description

Represents the address of a particular character in a `MuPDFStructuredTextPage`, in terms of block index, line index and character index.

Definition at line 961 of file `MuPDFStructuredTextPage.cs`.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 `MuPDFStructuredTextAddress()`

```
MuPDFCore.MuPDFStructuredTextAddress.MuPDFStructuredTextAddress (
    int blockIndex,
    int lineIndex,
    int characterIndex )
```

Creates a new `MuPDFStructuredTextAddress` from the specified indices.

Parameters

<code>blockIndex</code>	The index of the block.
<code>lineIndex</code>	The index of the line within the block.
<code>characterIndex</code>	The index of the character within the line.

Definition at line 984 of file [MuPDFStructuredTextPage.cs](#).

7.12.3 Member Function Documentation

7.12.3.1 CompareTo()

```
int MuPDFCore.MuPDFStructuredTextAddress.CompareTo (
    MuPDFStructuredTextAddress other )
```

Compares this [MuPDFStructuredTextAddress](#) with another [MuPDFStructuredTextAddress](#).

Parameters

<i>other</i>	The MuPDFStructuredTextAddress to compare with the current instance.
--------------	--

Returns

-1 if the *other* [MuPDFStructuredTextAddress](#) comes after the current instance, 1 if it comes before, or 0 if they represent the same address.

Definition at line 996 of file [MuPDFStructuredTextPage.cs](#).

7.12.3.2 Equals() [1/2]

```
bool MuPDFCore.MuPDFStructuredTextAddress.Equals (
    MuPDFStructuredTextAddress other )
```

Compares the current [MuPDFStructuredTextAddress](#) with another [MuPDFStructuredTextAddress](#).

Parameters

<i>other</i>	The other MuPDFStructuredTextAddress to compare with the current instance.
--------------	--

Returns

`true` if the two [MuPDFStructuredTextAddress](#)es represent the same address; otherwise, `false`.

Definition at line 1239 of file [MuPDFStructuredTextPage.cs](#).

7.12.3.3 Equals() [2/2]

```
override bool MuPDFCore.MuPDFStructuredTextAddress.Equals (
    object other )
```

Definition at line 1245 of file [MuPDFStructuredTextPage.cs](#).

7.12.3.4 GetHashCode()

```
override int MuPDFCore.MuPDFStructuredTextAddress.GetHashCode ( )
```

Definition at line 1195 of file [MuPDFStructuredTextPage.cs](#).

7.12.3.5 Increment()

```
MuPDFStructuredTextAddress? MuPDFCore.MuPDFStructuredTextAddress.Increment (
    MuPDFStructuredTextPage page )
```

Returns a [MuPDFStructuredTextAddress](#) corresponding to the next character in the specified page.

Parameters

<i>page</i>	The page the address refers to.
-------------	---------------------------------

Returns

A [MuPDFStructuredTextAddress](#) corresponding to the next character in the specified page.

Definition at line 1208 of file [MuPDFStructuredTextPage.cs](#).

7.12.3.6 operator"!=()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator!= (
    MuPDFStructuredTextAddress first,
    MuPDFStructuredTextAddress second ) [static]
```

Compares two [MuPDFStructuredTextAddress](#).

Parameters

<i>first</i>	The first MuPDFStructuredTextAddress to compare.
<i>second</i>	The second MuPDFStructuredTextAddress to compare.

Returns

true if the two MuPDFStructuredTextAddresses represent different addresses; otherwise, false.

Definition at line 1189 of file MuPDFStructuredTextPage.cs.

7.12.3.7 operator<()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator< (
    MuPDFStructuredTextAddress first,
    MuPDFStructuredTextAddress second ) [static]
```

Compares two MuPDFStructuredTextAddress.

Parameters

<i>first</i>	The first MuPDFStructuredTextAddress to compare.
<i>second</i>	The second MuPDFStructuredTextAddress to compare.

Returns

true if the *first* MuPDFStructuredTextAddress comes before the *second* one; otherwise, false.

Definition at line 1098 of file MuPDFStructuredTextPage.cs.

7.12.3.8 operator<=()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator<= (
    MuPDFStructuredTextAddress first,
    MuPDFStructuredTextAddress second ) [static]
```

Compares two MuPDFStructuredTextAddress.

Parameters

<i>first</i>	The first MuPDFStructuredTextAddress to compare.
<i>second</i>	The second MuPDFStructuredTextAddress to compare.

Returns

true if the *first* MuPDFStructuredTextAddress comes before the *second* one or if they represent the same address; otherwise, false.

Definition at line 1138 of file MuPDFStructuredTextPage.cs.

7.12.3.9 operator==()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator== (
    MuPDFStructuredTextAddress first,
    MuPDFStructuredTextAddress second ) [static]
```

Compares two [MuPDFStructuredTextAddress](#).

Parameters

<i>first</i>	The first MuPDFStructuredTextAddress to compare.
<i>second</i>	The second MuPDFStructuredTextAddress to compare.

Returns

true if the two [MuPDFStructuredTextAddress](#)es represent the same address; otherwise, false.

Definition at line 1178 of file [MuPDFStructuredTextPage.cs](#).

7.12.3.10 operator>()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator> (
    MuPDFStructuredTextAddress first,
    MuPDFStructuredTextAddress second ) [static]
```

Compares two [MuPDFStructuredTextAddress](#).

Parameters

<i>first</i>	The first MuPDFStructuredTextAddress to compare.
<i>second</i>	The second MuPDFStructuredTextAddress to compare.

Returns

true if the *first* [MuPDFStructuredTextAddress](#) comes after the *second* one; otherwise, false.

Definition at line 1018 of file [MuPDFStructuredTextPage.cs](#).

7.12.3.11 operator>=()

```
static bool MuPDFCore.MuPDFStructuredTextAddress.operator>= (
    MuPDFStructuredTextAddress first,
    MuPDFStructuredTextAddress second ) [static]
```

Compares two [MuPDFStructuredTextAddress](#).

Parameters

<i>first</i>	The first MuPDFStructuredTextAddress to compare.
<i>second</i>	The second MuPDFStructuredTextAddress to compare.

Returns

true if the *first* [MuPDFStructuredTextAddress](#) comes after the *second* one or if they represent the same address; otherwise, false.

Definition at line 1058 of file [MuPDFStructuredTextPage.cs](#).

7.12.4 Member Data Documentation

7.12.4.1 BlockIndex

```
readonly int MuPDFCore.MuPDFStructuredTextAddress.BlockIndex
```

The index of the block.

Definition at line 966 of file [MuPDFStructuredTextPage.cs](#).

7.12.4.2 CharacterIndex

```
readonly int MuPDFCore.MuPDFStructuredTextAddress.CharacterIndex
```

The index of the character within the line.

Definition at line 976 of file [MuPDFStructuredTextPage.cs](#).

7.12.4.3 LineIndex

```
readonly int MuPDFCore.MuPDFStructuredTextAddress.LineIndex
```

The index of the line within the block.

Definition at line 971 of file [MuPDFStructuredTextPage.cs](#).

The documentation for this struct was generated from the following file:

- [MuPDFCore/MuPDFStructuredTextPage.cs](#)

7.13 MuPDFCore.MuPDFStructuredTextAddressSpan Class Reference

Represents a range of characters in a [MuPDFStructuredTextPage](#).

Public Member Functions

- [MuPDFStructuredTextAddressSpan](#) ([MuPDFStructuredTextAddress](#) start, [MuPDFStructuredTextAddress?](#) end)

Creates a new [MuPDFStructuredTextAddressSpan](#) corresponding to the specified character range.

Public Attributes

- readonly [MuPDFStructuredTextAddress Start](#)
The address of the start of the range.
- readonly? [MuPDFStructuredTextAddress End](#)
The address of the end of the range (inclusive), or null to signify an empty range.

7.13.1 Detailed Description

Represents a range of characters in a [MuPDFStructuredTextPage](#).

Definition at line 1254 of file [MuPDFStructuredTextPage.cs](#).

7.13.2 Constructor & Destructor Documentation

7.13.2.1 MuPDFStructuredTextAddressSpan()

```
MuPDFCore.MuPDFStructuredTextAddressSpan.MuPDFStructuredTextAddressSpan (
    MuPDFStructuredTextAddress start,
    MuPDFStructuredTextAddress? end )
```

Creates a new [MuPDFStructuredTextAddressSpan](#) corresponding to the specified character range.

Parameters

<code>start</code>	The address of the start of the range.
<code>end</code>	The address of the end of the range (inclusive), or null to signify an empty range.

Definition at line 1271 of file [MuPDFStructuredTextPage.cs](#).

7.13.3 Member Data Documentation

7.13.3.1 End

```
readonly? MuPDFStructuredTextAddress MuPDFCore.MuPDFStructuredTextAddressSpan.End
```

The address of the end of the range (inclusive), or null to signify an empty range.

Definition at line 1264 of file [MuPDFStructuredTextPage.cs](#).

7.13.3.2 Start

```
readonly MuPDFStructuredTextAddress MuPDFCore.MuPDFStructuredTextAddressSpan.Start
```

The address of the start of the range.

Definition at line 1259 of file [MuPDFStructuredTextPage.cs](#).

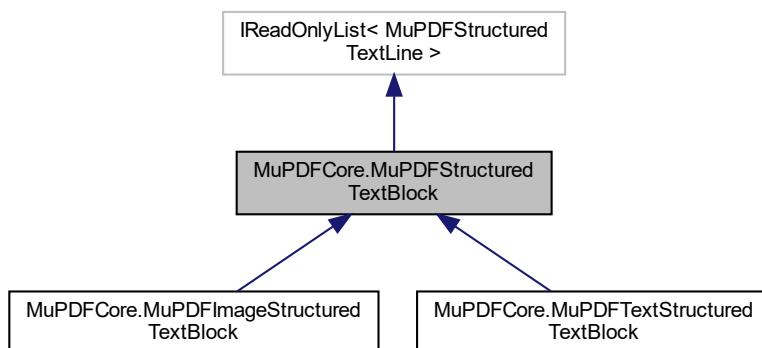
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFStructuredTextPage.cs](#)

7.14 MuPDFCore.MuPDFStructuredTextBlock Class Reference

Represents a structured text block containing text or an image.

Inheritance diagram for MuPDFCore.MuPDFStructuredTextBlock:



Public Types

- enum [Types](#) { [Text](#) = 0 , [Image](#) = 1 }

Defines the type of the block.

Public Member Functions

- abstract `IEnumerator< MuPDFStructuredTextLine > GetEnumerator ()`

Properties

- abstract `Types Type [get]`
The type of the block.
- `Rectangle BoundingBox [get]`
The bounding box of the block.
- abstract `int Count [get]`
The number of lines in the block.
- abstract `MuPDFStructuredTextLine this[int index] [get]`
Gets the specified line from the block.

7.14.1 Detailed Description

Represents a structured text block containing text or an image.

Definition at line 556 of file [MuPDFStructuredTextPage.cs](#).

7.14.2 Member Enumeration Documentation

7.14.2.1 Types

```
enum MuPDFCore.MuPDFStructuredTextBlock.Types
```

Defines the type of the block.

Enumerator

Text	The block contains text.
Image	The block contains an image.

Definition at line 561 of file [MuPDFStructuredTextPage.cs](#).

7.14.3 Member Function Documentation

7.14.3.1 GetEnumerator()

```
abstract IEnumerator< MuPDFStructuredTextLine > MuPDFCore.MuPDFStructuredTextBlock.GetEnumerator()
( ) [pure virtual]
```

Implemented in [MuPDFCore.MuPDFImageStructuredTextBlock](#), and [MuPDFCore.MuPDFTextStructuredTextBlock](#).

7.14.4 Property Documentation

7.14.4.1 BoundingBox

```
Rectangle MuPDFCore.MuPDFStructuredTextBlock.BoundingBox [get]
```

The bounding box of the block.

Definition at line 582 of file [MuPDFStructuredTextPage.cs](#).

7.14.4.2 Count

```
abstract int MuPDFCore.MuPDFStructuredTextBlock.Count [get]
```

The number of lines in the block.

Definition at line 587 of file [MuPDFStructuredTextPage.cs](#).

7.14.4.3 this[int index]

```
abstract MuPDFStructuredTextLine MuPDFCore.MuPDFStructuredTextBlock.this[int index] [get]
```

Gets the specified line from the block.

Parameters

<i>index</i>	The index of the line to extract.
--------------	-----------------------------------

Returns

The [MuPDFStructuredTextLine](#) with the specified *index*.

Definition at line 594 of file [MuPDFStructuredTextPage.cs](#).

7.14.4.4 Type

```
abstract Types MuPDFCore.MuPDFStructuredTextBlock.Type [get]
```

The type of the block.

Definition at line 577 of file [MuPDFStructuredTextPage.cs](#).

The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.15 MuPDFCore.MuPDFStructuredTextCharacter Class Reference

Represents a single text character.

Public Member Functions

- override string [ToString \(\)](#)
Returns a string representation of the character.

Properties

- int [CodePoint \[get\]](#)
The unicode code point of the character.
- string [Character \[get\]](#)
A string representation of the character. It may consist of a single char or of a surrogate pair of chars.
- int [Color \[get\]](#)
An sRGB hex representation of the colour of the character.
- [PointF Origin \[get\]](#)
The baseline origin of the character.
- [Quad BoundingQuad \[get\]](#)
A quadrilateral bound for the character. This may or may not be a rectangle.
- float [Size \[get\]](#)
The size in points of the character.

7.15.1 Detailed Description

Represents a single text character.

Definition at line [906](#) of file [MuPDFStructuredTextPage.cs](#).

7.15.2 Member Function Documentation

7.15.2.1 ToString()

```
override string MuPDFCore.MuPDFStructuredTextCharacter.ToString ( )
```

Returns a string representation of the character.

Returns

A string representation of the character.

Definition at line [952](#) of file [MuPDFStructuredTextPage.cs](#).

7.15.3 Property Documentation

7.15.3.1 BoundingQuad

```
Quad MuPDFCore.MuPDFStructuredTextCharacter.BoundingQuad [get]
```

A quadrilateral bound for the character. This may or may not be a rectangle.

Definition at line 931 of file [MuPDFStructuredTextPage.cs](#).

7.15.3.2 Character

```
string MuPDFCore.MuPDFStructuredTextCharacter.Character [get]
```

A string representation of the character. It may consist of a single char or of a surrogate pair of chars.

Definition at line 916 of file [MuPDFStructuredTextPage.cs](#).

7.15.3.3 CodePoint

```
int MuPDFCore.MuPDFStructuredTextCharacter.CodePoint [get]
```

The unicode code point of the character.

Definition at line 911 of file [MuPDFStructuredTextPage.cs](#).

7.15.3.4 Color

```
int MuPDFCore.MuPDFStructuredTextCharacter.Color [get]
```

An sRGB hex representation of the colour of the character.

Definition at line 921 of file [MuPDFStructuredTextPage.cs](#).

7.15.3.5 Origin

```
PointF MuPDFCore.MuPDFStructuredTextCharacter.Origin [get]
```

The baseline origin of the character.

Definition at line 926 of file [MuPDFStructuredTextPage.cs](#).

7.15.3.6 Size

```
float MuPDFCore.MuPDFStructuredTextCharacter.Size [get]
```

The size in points of the character.

Definition at line 936 of file [MuPDFStructuredTextPage.cs](#).

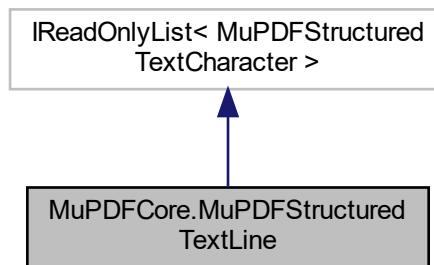
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFStructuredTextPage.cs](#)

7.16 MuPDFCore.MuPDFStructuredTextLine Class Reference

Represents a single line of text (i.e. characters that share a common baseline).

Inheritance diagram for MuPDFCore.MuPDFStructuredTextLine:



Public Types

- enum [WritingModes](#) { `Horizontal` = 0 , `Vertical` = 1 }

Defines the writing mode of the text.

Public Member Functions

- override string [ToString](#) ()
Returns a string representation of the line.
- `IEnumerator< MuPDFStructuredTextCharacter > GetEnumerator ()`

Properties

- **WritingModes WritingMode** [get]
The writing mode of the text.
- **PointF Direction** [get]
The normalised direction of the text baseline.
- **Rectangle BoundingBox** [get]
The bounding box of the line.
- **MuPDFStructuredTextCharacter[] Characters** [get]
The characters contained in the line.
- **string Text** [get]
A string representation of the characters contained in the line.
- **int Count** [get]
The number of characters in the line.
- **MuPDFStructuredTextCharacter this[int index]** [get]
Gets the specified character from the line.

7.16.1 Detailed Description

Represents a single line of text (i.e. characters that share a common baseline).

Definition at line [752](#) of file [MuPDFStructuredTextPage.cs](#).

7.16.2 Member Enumeration Documentation

7.16.2.1 WritingModes

enum [MuPDFCore.MuPDFStructuredTextLine.WritingModes](#)

Defines the writing mode of the text.

Enumerator

Horizontal	The text is written horizontally.
Vertical	The text is written vertically.

Definition at line [757](#) of file [MuPDFStructuredTextPage.cs](#).

7.16.3 Member Function Documentation

7.16.3.1 GetEnumerator()

```
IEnumerator< MuPDFStructuredTextCharacter > MuPDFCore.MuPDFStructuredTextLine.GetEnumerator ( )
```

Definition at line 892 of file [MuPDFStructuredTextPage.cs](#).

7.16.3.2 ToString()

```
override string MuPDFCore.MuPDFStructuredTextLine.ToString ( )
```

Returns a string representation of the line.

Returns

A string representation of the line.

Definition at line 886 of file [MuPDFStructuredTextPage.cs](#).

7.16.4 Property Documentation

7.16.4.1 BoundingBox

```
Rectangle MuPDFCore.MuPDFStructuredTextLine.BoundingBox [get]
```

The bounding box of the line.

Definition at line 783 of file [MuPDFStructuredTextPage.cs](#).

7.16.4.2 Characters

```
MuPDFStructuredTextCharacter [ ] MuPDFCore.MuPDFStructuredTextLine.Characters [get]
```

The characters contained in the line.

Definition at line 788 of file [MuPDFStructuredTextPage.cs](#).

7.16.4.3 Count

```
int MuPDFCore.MuPDFStructuredTextLine.Count [get]
```

The number of characters in the line.

Definition at line 798 of file [MuPDFStructuredTextPage.cs](#).

7.16.4.4 Direction

```
PointF MuPDFCore.MuPDFStructuredTextLine.Direction [get]
```

The normalised direction of the text baseline.

Definition at line 778 of file [MuPDFStructuredTextPage.cs](#).

7.16.4.5 Text

```
string MuPDFCore.MuPDFStructuredTextLine.Text [get]
```

A string representation of the characters contained in the line.

Definition at line 793 of file [MuPDFStructuredTextPage.cs](#).

7.16.4.6 this[int index]

```
MuPDFStructuredTextCharacter MuPDFCore.MuPDFStructuredTextLine.this[int index] [get]
```

Gets the specified character from the line.

Parameters

<i>index</i>	The index of the character.
--------------	-----------------------------

Returns

The [MuPDFStructuredTextCharacter](#) with the specified *index*.

Definition at line 805 of file [MuPDFStructuredTextPage.cs](#).

7.16.4.7 WritingMode

`WritingModes` `MuPDFCore.MuPDFStructuredTextLine.WritingMode` [get]

The writing mode of the text.

Definition at line 773 of file [MuPDFStructuredTextPage.cs](#).

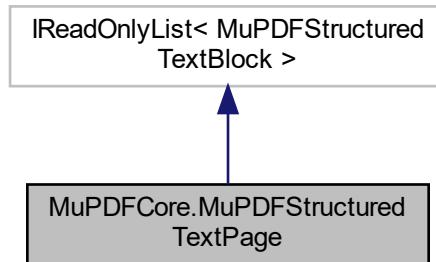
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFStructuredTextPage.cs](#)

7.17 MuPDFCore.MuPDFStructuredTextPage Class Reference

Represents a structured representation of the text contained in a page.

Inheritance diagram for `MuPDFCore.MuPDFStructuredTextPage`:



Public Member Functions

- `MuPDFStructuredTextAddress? GetHitAddress (PointF point, bool includeImages)`
Gets the address of the character that contains the specified point in page units.
- `MuPDFStructuredTextAddress? GetClosestHitAddress (PointF point, bool includeImages)`
Gets the address of the character that contains the specified point in page units.
- `IEnumerable< Quad > GetHighlightQuads (MuPDFStructuredTextAddressSpan range, bool includeImages)`
Gets a collection of `Quads` delimiting the specified character range . Where possible, these are collapsed at the line and block level. Each `Quad` may or may not be a rectangle.
- `string GetText (MuPDFStructuredTextAddressSpan range)`
Gets the text corresponding to the specified character range . Blocks containing images are ignored.
- `IEnumerable< MuPDFStructuredTextAddressSpan > Search (Regex needle)`
Searches for the specified `Regex` in the text of the page. A single match cannot span multiple lines.
- `IEnumerator< MuPDFStructuredTextBlock > GetEnumerator ()`

Properties

- `MuPDFStructuredTextBlock[] StructuredTextBlocks [get]`
The blocks contained in the page.
- `int Count [get]`
The number of blocks in the page.
- `MuPDFStructuredTextBlock this[int index] [get]`
Gets the specified block in the page.
- `MuPDFStructuredTextCharacter this[MuPDFStructuredTextAddress address] [get]`
Gets the specified character in the page.

7.17.1 Detailed Description

Represents a structured representation of the text contained in a page.

Definition at line 31 of file [MuPDFStructuredTextPage.cs](#).

7.17.2 Member Function Documentation

7.17.2.1 GetClosestHitAddress()

```
MuPDFStructuredTextAddress? MuPDFCore.MuPDFStructuredTextPage.GetClosestHitAddress (
    PointF point,
    bool includeImages )
```

Gets the address of the character that contains the specified *point* in page units.

Parameters

<i>point</i>	The point that must be closest to the character. This is expressed in page units (i.e. with a zoom factor of 1).
<i>includeImages</i>	If this is <code>true</code> , blocks containing images may be returned. Otherwise, only blocks containing text are considered.

Returns

The address of the character closest to the specified *point*. This is `null` only if the page contains no characters.

Definition at line 208 of file [MuPDFStructuredTextPage.cs](#).

7.17.2.2 GetEnumerator()

```
IEnumerator< MuPDFStructuredTextBlock > MuPDFCore.MuPDFStructuredTextPage.GetEnumerator ()
```

Definition at line 542 of file [MuPDFStructuredTextPage.cs](#).

7.17.2.3 GetHighlightQuads()

```
IEnumerable< Quad > MuPDFCore.MuPDFStructuredTextPage.GetHighlightQuads (
    MuPDFStructuredTextAddressSpan range,
    bool includeImages )
```

Gets a collection of [Quads](#) delimiting the specified character *range*. Where possible, these are collapsed at the line and block level. Each [Quad](#) may or may not be a rectangle.

Parameters

<i>range</i>	A MuPDFStructuredTextAddressSpan representing the character range
<i>includeImages</i>	If this is <code>true</code> , the bounding boxes for blocks containing images are also returned. Otherwise, only blocks containing text are considered.

Returns

A lazy collection of [Quads](#) delimiting the characters in the specified *includeImages*.

Definition at line 281 of file [MuPDFStructuredTextPage.cs](#).

7.17.2.4 GetHitAddress()

```
MuPDFStructuredTextAddress? MuPDFCore.MuPDFStructuredTextPage.GetHitAddress (
    PointF point,
    bool includeImages )
```

Gets the address of the character that contains the specified *point* in page units.

Parameters

<i>point</i>	The point that must be contained by the character. This is expressed in page units (i.e. with a zoom factor of 1).
<i>includeImages</i>	If this is <code>true</code> , blocks containing images may be returned. Otherwise, only blocks containing text are considered.

Returns

The address of the character containing the specified *point*, or `null` if no character contains the *point*.

Definition at line 174 of file [MuPDFStructuredTextPage.cs](#).

7.17.2.5 GetText()

```
string MuPDFCore.MuPDFStructuredTextPage.GetText (
    MuPDFStructuredTextAddressSpan range )
```

Gets the text corresponding to the specified character *range*. Blocks containing images are ignored.

Parameters

<i>range</i>	A MuPDFStructuredTextAddressSpan representing the range of text to extract.
--------------	---

Returns

A string representation of the text contained in the specified *range*.

Definition at line 386 of file [MuPDFStructuredTextPage.cs](#).

7.17.2.6 Search()

```
IEnumerable< MuPDFStructuredTextAddressSpan > MuPDFCore.MuPDFStructuredTextPage.Search (
    Regex needle )
```

Searches for the specified Regex in the text of the page. A single match cannot span multiple lines.

Parameters

<i>needle</i>	The Regex to search for.
---------------	--------------------------

Returns

A lazy collection of [MuPDFStructuredTextAddressSpan](#)s representing all the occurrences of the *needle* in the text.

Definition at line 497 of file [MuPDFStructuredTextPage.cs](#).

7.17.3 Property Documentation

7.17.3.1 Count

```
int MuPDFCore.MuPDFStructuredTextPage.Count [get]
```

The number of blocks in the page.

Definition at line 41 of file [MuPDFStructuredTextPage.cs](#).

7.17.3.2 StructuredTextBlocks

```
MuPDFStructuredTextBlock [] MuPDFCore.MuPDFStructuredTextPage.StructuredTextBlocks [get]
```

The blocks contained in the page.

Definition at line 36 of file [MuPDFStructuredTextPage.cs](#).

7.17.3.3 this[int index]

```
MuPDFStructuredTextBlock MuPDFCore.MuPDFStructuredTextPage.this[int index] [get]
```

Gets the specified block in the page.

Parameters

<i>index</i>	The index of the block.
--------------	-------------------------

Returns

The block with the specified *index*.

Definition at line 48 of file [MuPDFStructuredTextPage.cs](#).

7.17.3.4 this[MuPDFStructuredTextAddress address]

```
MuPDFStructuredTextCharacter MuPDFCore.MuPDFStructuredTextPage.this[MuPDFStructuredTextAddress address] [get]
```

Gets the specified character in the page.

Parameters

<i>address</i>	The address (block, line and character index) of the character.
----------------	---

Returns

A [MuPDFStructuredTextCharacter](#) representing the specified character.

Definition at line 55 of file [MuPDFStructuredTextPage.cs](#).

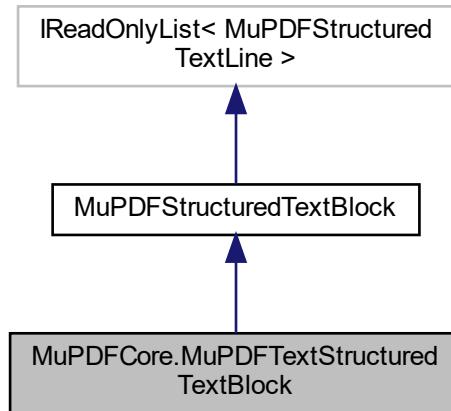
The documentation for this class was generated from the following file:

- MuPDFCore/MuPDFStructuredTextPage.cs

7.18 MuPDFCore.MuPDFTextStructuredTextBlock Class Reference

Represents a block containing multiple lines of text (typically a paragraph).

Inheritance diagram for MuPDFCore.MuPDFTextStructuredTextBlock:



Public Member Functions

- override `IEnumerator< MuPDFStructuredTextLine > GetEnumerator ()`
- override `string ToString ()`

Returns the text contained in the block as a string.

Properties

- override `Types Type [get]`
- `MuPDFStructuredTextLine[] Lines [get]`
The lines of text in the block.
- override `int Count [get]`
- override `MuPDFStructuredTextLine this[int index] [get]`

Additional Inherited Members

7.18.1 Detailed Description

Represents a block containing multiple lines of text (typically a paragraph).

Definition at line 656 of file [MuPDFStructuredTextPage.cs](#).

7.18.2 Member Function Documentation

7.18.2.1 GetEnumerator()

```
override IEnumrator< MuPDFStructuredTextLine > MuPDFCore.MuPDFTextStructuredTextBlock.GetEnumerator ( ) [virtual]
```

Implements [MuPDFCore.MuPDFTextBlock](#).

Definition at line 727 of file [MuPDFStructuredTextPage.cs](#).

7.18.2.2 ToString()

```
override string MuPDFCore.MuPDFTextStructuredTextBlock.ToString ( )
```

Returns the text contained in the block as a string.

Returns

The text contained in the block as a string. If the block contains at least one line, the return value has a line terminator at the end.

Definition at line 736 of file [MuPDFStructuredTextPage.cs](#).

7.18.3 Property Documentation

7.18.3.1 Count

```
override int MuPDFCore.MuPDFTextStructuredTextBlock.Count [get]
```

Definition at line 667 of file [MuPDFStructuredTextPage.cs](#).

7.18.3.2 Lines

```
MuPDFStructuredTextLine [ ] MuPDFCore.MuPDFTextStructuredTextBlock.Lines [get]
```

The lines of text in the block.

Definition at line 664 of file [MuPDFStructuredTextPage.cs](#).

7.18.3.3 `this[int index]`

```
override MuPDFStructuredTextLine MuPDFCore.MuPDFTextStructuredTextBlock.this[int index] [get]
```

Definition at line 670 of file [MuPDFStructuredTextPage.cs](#).

7.18.3.4 `Type`

```
override Types MuPDFCore.MuPDFTextStructuredTextBlock.Type [get]
```

Definition at line 659 of file [MuPDFStructuredTextPage.cs](#).

The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFStructuredTextPage.cs](#)

7.19 MuPDFCore.OCRProgressInfo Class Reference

Describes OCR progress.

Properties

- double `Progress` [get]
A value between 0 and 1, indicating how much progress has been completed.

7.19.1 Detailed Description

Describes OCR progress.

Definition at line 15 of file [MuPDFStructuredTextPage.cs](#).

7.19.2 Property Documentation

7.19.2.1 Progress

```
double MuPDFCore.OCRProgressInfo.Progress [get]
```

A value between 0 and 1, indicating how much progress has been completed.

Definition at line 20 of file [MuPDFStructuredTextPage.cs](#).

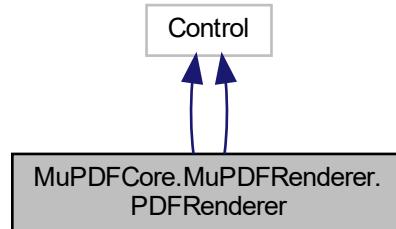
The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDFStructuredTextPage.cs](#)

7.20 MuPDFCore.MuPDFRenderer.PDFRenderer Class Reference

A control to render PDF documents (and other formats), potentially using multiple threads.

Inheritance diagram for MuPDFCore.MuPDFRenderer.PDFRenderer:



Public Types

- enum [PointerEventHandlers](#) { [Pan](#) , [Highlight](#) , [PanHighlight](#) , [Custom](#) }
- Identifies the action to perform on pointer events.*

Public Member Functions

- [PDFRenderer \(\)](#)
Initializes a new instance of the [PDFRenderer](#) class.
- void [Initialize \(\[MuPDFDocument\]\(#\) document, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, \[TesseractLanguage\]\(#\) ocrLanguage=null\)](#)
Set up the [PDFRenderer](#) to display a page of a [MuPDFDocument](#).
- async Task [InitializeAsync \(\[MuPDFDocument\]\(#\) document, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, \[TesseractLanguage\]\(#\) ocrLanguage=null, CancellationToken ocrCancellationToken=default, IProgress<\[OCRProgressInfo\]\(#\)> ocrProgress=null\)](#)
Set up the [PDFRenderer](#) to display a page of a [MuPDFDocument](#). The OCR step is run asynchronously, in order not to block the UI thread.
- void [Initialize \(string fileName, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, \[TesseractLanguage\]\(#\) ocrLanguage=null\)](#)
Set up the [PDFRenderer](#) to display a page of a document that will be loaded from disk.
- async Task [InitializeAsync \(string fileName, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, \[TesseractLanguage\]\(#\) ocrLanguage=null, CancellationToken ocrCancellationToken=default, IProgress<\[OCRProgressInfo\]\(#\)> ocrProgress=null\)](#)
Set up the [PDFRenderer](#) to display a page of a document that will be loaded from disk. The OCR step is run asynchronously, in order not to block the UI thread.
- void [Initialize \(\[MemoryStream\]\(#\) ms, \[InputFileTypes\]\(#\) fileType, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, \[TesseractLanguage\]\(#\) ocrLanguage=null\)](#)
Set up the [PDFRenderer](#) to display a page of a document that will be loaded from a [MemoryStream](#).
- async Task [InitializeAsync \(\[MemoryStream\]\(#\) ms, \[InputFileTypes\]\(#\) fileType, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, \[TesseractLanguage\]\(#\) ocrLanguage=null, CancellationToken ocrCancellationToken=default, IProgress<\[OCRProgressInfo\]\(#\)> ocrProgress=null\)](#)
Set up the [PDFRenderer](#) to display a page of a document that will be loaded from a [MemoryStream](#).

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from a [MemoryStream](#). The OCR step is run asynchronously, in order not to block the UI thread.

- void [Initialize](#) (byte[] dataBytes, [InputFileTypes](#) fileType, int offset=0, int length=-1, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, [TesseractLanguage](#) ocrLanguage=null)

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from an array of bytes.

- async Task [InitializeAsync](#) (byte[] dataBytes, [InputFileTypes](#) fileType, int offset=0, int length=-1, int threadCount=0, int pageNumber=0, double resolutionMultiplier=1, bool includeAnnotations=true, [TesseractLanguage](#) ocrLanguage=null, CancellationToken ocrCancellationToken=default, IProgress<[OCRProgressInfo](#)> ocrProgress=null)

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from an array of bytes. The OCR step is run asynchronously, in order not to block the UI thread.

- void [ReleaseResources](#) ()

Release resources held by this [PDFRenderer](#). This is not an irreversible step: using one of the [Initialize](#) overloads after calling this method will restore functionality.

- void [SetDisplayAreaNow](#) (Rect value)

Set the current display area to the specified value , skipping all transitions.

- void [ZoomStep](#) (double count, Point? center=null)

Zoom around a point.

- void [Contain](#) ()

Alter the display area so that the whole page fits on screen.

- void [Cover](#) ()

Alter the display area so that the page covers the whole surface of the [PDFRenderer](#) (even though parts of the page may be outside it).

- [RenderProgress](#) [GetProgress](#) ()

Get the current rendering progress.

- string [GetSelectedText](#) ()

Get the currently selected text.

- void [SelectAll](#) ()

Selects all the text in the document.

- int [Search](#) (Regex needle)

Highlights all matches of the specified Regex in the text and returns the number of matches found. Matches cannot span multiple lines.

- override void [Render](#) (DrawingContext context)

Draw the rendered document.

Static Public Attributes

- static readonly DirectProperty< [PDFRenderer](#), int > [RenderThreadCountProperty](#) = AvaloniaProperty.RegisterDirect<[PDFRenderer](#), int>(nameof([RenderThreadCount](#)), o => o.RenderThreadCount)

Defines the [RenderThreadCount](#) property.
- static readonly DirectProperty< [PDFRenderer](#), int > [PageNumberProperty](#) = AvaloniaProperty.RegisterDirect<[PDFRenderer](#), int>(nameof([PageNumber](#)), o => o.PageNumber)

Defines the [PageNumber](#) property.
- static readonly DirectProperty< [PDFRenderer](#), bool > [IsViewerInitializedProperty](#) = AvaloniaProperty.RegisterDirect<[PDFRenderer](#), bool>(nameof([IsViewerInitialized](#)), o => o.IsViewerInitialized)

Defines the [IsViewerInitialized](#) property.
- static readonly DirectProperty< [PDFRenderer](#), Rect > [PageSizeProperty](#) = AvaloniaProperty.RegisterDirect<[PDFRenderer](#), Rect>(nameof([PageSize](#)), o => o.PageSize)

Defines the [PageSize](#) property.
- static readonly StyledProperty< Rect > [DisplayAreaProperty](#) = AvaloniaProperty.Register<[PDFRenderer](#), Rect>(nameof([DisplayArea](#)))

- static readonly StyledProperty< double > **ZoomIncrementProperty** = AvaloniaProperty.Register<PDFRenderer, double>(nameof(ZoomIncrement), Math.Pow(2, 1.0 / 3.0), defaultBindingMode: Avalonia.Data.BindingMode.TwoWay)

Defines the [ZoomIncrement](#) property.
- static readonly StyledProperty< IBrush > **BackgroundProperty** = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(Background))

Defines the [Background](#) property.
- static readonly StyledProperty< IBrush > **PageBackgroundProperty** = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(PageBackground))

Defines the [PageBackground](#) property.
- static readonly DirectProperty< PDFRenderer, double > **ZoomProperty** = AvaloniaProperty.Register<PDFRenderer, Direct>(nameof(Zoom), o => o.Zoom, (o, v) => o.Zoom = v, defaultBindingMode: Avalonia.Data.BindingMode.TwoWay)

Defines the [Zoom](#) property.
- static readonly StyledProperty< PointerEventHandlers > **PointerEventHandlerTypeProperty** = AvaloniaProperty.Register<PDFRenderer, PointerEventHandlers>(nameof(PointerEventHandlersType), PointerEventHandlers.PanHigh)

Defines the [PointerEventHandlersType](#) property.
- static readonly StyledProperty< bool > **ZoomEnabledProperty** = AvaloniaProperty.Register<PDFRenderer, bool>(nameof(ZoomEnabled), true)

Defines the [ZoomEnabled](#) property.
- static readonly StyledProperty< MuPDFStructuredTextAddressSpan > **SelectionProperty** = AvaloniaProperty.Register<PDFRenderer, MuPDFStructuredTextAddressSpan>(nameof(Selection), null)

Defines the [Selection](#) property.
- static readonly StyledProperty< IBrush > **SelectionBrushProperty** = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(SelectionBrush), new SolidColorBrush(Color.FromArgb(96, 86, 180, 233)))

Defines the [SelectionBrush](#) property.
- static readonly StyledProperty< IEnumerable< MuPDFStructuredTextAddressSpan > > **HighlightedRegionsProperty** = AvaloniaProperty.Register<PDFRenderer, IEnumerable<MuPDFStructuredTextAddressSpan>>(nameof(HighlightedRegions), null)

Defines the [HighlightedRegions](#) property.
- static readonly StyledProperty< IBrush > **HighlightBrushProperty** = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(HighlightBrush), new SolidColorBrush(Color.FromArgb(96, 230, 159, 0)))

Defines the [HighlightBrush](#) property.

Properties

- int **RenderThreadCount** [get]

Exposes the number of threads that the current instance is using to render the document. Read-only.
- int **PageNumber** [get]

Exposes the number of the page that the current instance is rendering. Read-only.
- bool **IsViewerInitialized** [get]

Whether the current instance has been initialised with a document to render or not. Read-only.
- Rect **PageSize** [get]

Exposes the size of the page that is drawn by the current instance (in page units).
- Rect **DisplayArea** [get, set]

The region of the page (in page units) that is currently displayed by the current instance. This always has the same aspect ratio of the bounds of this control. When this is set, the value is sanitised so that the smallest rectangle with the correct aspect ratio containing the requested value is chosen.
- double **ZoomIncrement** [get, set]

Determines by how much the scale will be increased/decreased by the [ZoomStep\(double, Point?\)](#) method. Set this to a value smaller than 1 to invert the zoom in/out direction.

- IBrush **Background** [get, set]

The background colour of the control.
- IBrush **PageBackground** [get, set]

The background colour to use for the page drawn by the control.
- double **Zoom** [get, set]

The current zoom level. Setting this will change the [DisplayArea](#) appropriately, zooming around the center of the [DisplayArea](#).
- PointerEventHandlers **PointerEventHandlersType** [get, set]

Whether the default handlers for pointer events (which are used for panning around the page) should be enabled. If this is false, you will have to implement your own way to pan around the document by changing the [DisplayArea](#).
- bool **ZoomEnabled** [get, set]

Whether the default handlers for pointer wheel events (which are used for zooming in/out) should be enabled. If this is false, you will have to implement your own way to zoom by changing the [DisplayArea](#).
- MuPDFStructuredTextAddressSpan **Selection** [get, set]

The start and end of the currently selected text.
- IBrush **SelectionBrush** [get, set]

The colour used to highlight the [Selection](#).
- IEnumerable< MuPDFStructuredTextAddressSpan > **HighlightedRegions** [get, set]

A collection of highlighted regions, e.g. as a result of a text search.
- IBrush **HighlightBrush** [get, set]

The colour used to highlight the [HighlightedRegions](#).

7.20.1 Detailed Description

A control to render PDF documents (and other formats), potentially using multiple threads.

Definition at line 42 of file [PDFRenderer.cs](#).

7.20.2 Member Enumeration Documentation

7.20.2.1 PointerEventHandlers

```
enum MuPDFCore.MuPDFRenderer.PDFRenderer.PointerEventHandlers
```

Identifies the action to perform on pointer events.

Enumerator

Pan	Pointer events will be used to pan around the page.
Highlight	Pointer events will be used to highlight text.
PanHighlight	Pointer events will be used to pan around the page or to highlight text, depending on where they start.
Custom	Pointer events will be ignored. If you use this value, you will have to implement your own way to pan around the document by changing the DisplayArea or to select text.

Definition at line 246 of file [PDFRenderer.Properties.cs](#).

7.20.3 Constructor & Destructor Documentation

7.20.3.1 PDFRenderer()

```
MuPDFCore.MuPDFRenderer.PDFRenderer.PDFRenderer ( )
```

Initializes a new instance of the [PDFRenderer](#) class.

Definition at line [203](#) of file [PDFRenderer.cs](#).

7.20.4 Member Function Documentation

7.20.4.1 Contain()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Contain ( )
```

Alter the display area so that the whole page fits on screen.

Definition at line [698](#) of file [PDFRenderer.cs](#).

7.20.4.2 Cover()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Cover ( )
```

Alter the display area so that the page covers the whole surface of the [PDFRenderer](#) (even though parts of the page may be outside it).

Definition at line [707](#) of file [PDFRenderer.cs](#).

7.20.4.3 GetProgress()

```
RenderProgress MuPDFCore.MuPDFRenderer.PDFRenderer.GetProgress ( )
```

Get the current rendering progress.

Returns

A [RenderProgress](#) object with information about the rendering progress of each thread.

Definition at line [728](#) of file [PDFRenderer.cs](#).

7.20.4.4 GetSelectedText()

```
string MuPDFCore.MuPDFRenderer.PDFRenderer.GetSelectedText ( )
```

Get the currently selected text.

Returns

The currently selected text.

Definition at line 737 of file [PDFRenderer.cs](#).

7.20.4.5 Initialize() [1/4]

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Initialize (
    byte[] dataBytes,
    InputFileTypes fileType,
    int offset = 0,
    int length = -1,
    int threadCount = 0,
    int pageNumber = 0,
    double resolutionMultiplier = 1,
    bool includeAnnotations = true,
    TesseractLanguage ocrLanguage = null )
```

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from an array of bytes.

Parameters

<i>dataBytes</i>	The bytes of the document that should be opened. The array will be copied and can be safely discarded/altered after this method returns.
<i>fileType</i>	The format of the document.
<i>offset</i>	The offset in the byte array at which the document starts.
<i>length</i>	The length of the document in bytes. If this is < 0, the whole array is used.
<i>threadCount</i>	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than <i>threadCount</i> that satisfies this condition is used.
<i>pageNumber</i>	The index of the page that should be rendered. The first page has index 0.
<i>resolutionMultiplier</i>	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If <i>resolutionMultiplier</i> is 1, the resolution will match the size (in screen units) of the PDFRenderer .
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is <code>null</code> , no OCR is performed.

Definition at line 403 of file [PDFRenderer.cs](#).

7.20.4.6 Initialize() [2/4]

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Initialize (
    MemoryStream ms,
    InputFileTypes fileType,
    int threadCount = 0,
    int pageNumber = 0,
    double resolutionMultiplier = 1,
    bool includeAnnotations = true,
    TesseractLanguage ocrLanguage = null )
```

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from a [MemoryStream](#).

Parameters

<i>ms</i>	The MemoryStream containing the document that should be opened. This can be safely disposed after this method returns.
<i>fileType</i>	The format of the document.
<i>threadCount</i>	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than <i>threadCount</i> that satisfies this condition is used.
<i>pageNumber</i>	The index of the page that should be rendered. The first page has index 0.
<i>resolutionMultiplier</i>	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If <i>resolutionMultiplier</i> is 1, the resolution will match the size (in screen units) of the PDFRenderer .
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is <code>null</code> , no OCR is performed.

Definition at line 359 of file [PDFRenderer.cs](#).

7.20.4.7 Initialize() [3/4]

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Initialize (
    MuPDFDocument document,
    int threadCount = 0,
    int pageNumber = 0,
    double resolutionMultiplier = 1,
    bool includeAnnotations = true,
    TesseractLanguage ocrLanguage = null )
```

Set up the [PDFRenderer](#) to display a page of a [MuPDFDocument](#).

Parameters

<i>document</i>	The MuPDFDocument to render.
<i>threadCount</i>	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than <i>threadCount</i> that satisfies this condition is used.

Parameters

<i>pageNumber</i>	The index of the page that should be rendered. The first page has index 0.
<i>resolutionMultiplier</i>	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If <i>resolutionMultiplier</i> is 1, the resolution will match the size (in screen units) of the PDFRenderer .
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.

Definition at line 258 of file [PDFRenderer.cs](#).

7.20.4.8 Initialize() [4/4]

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.Initialize (
    string fileName,
    int threadCount = 0,
    int pageNumber = 0,
    double resolutionMultiplier = 1,
    bool includeAnnotations = true,
    TesseractLanguage ocrLanguage = null )
```

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from disk.

Parameters

<i>fileName</i>	The path to the document that should be opened.
<i>threadCount</i>	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than <i>threadCount</i> that satisfies this condition is used.
<i>pageNumber</i>	The index of the page that should be rendered. The first page has index 0.
<i>resolutionMultiplier</i>	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If <i>resolutionMultiplier</i> is 1, the resolution will match the size (in screen units) of the PDFRenderer .
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.

Definition at line 308 of file [PDFRenderer.cs](#).

7.20.4.9 InitializeAsync() [1/4]

```
async Task MuPDFCore.MuPDFRenderer.PDFRenderer.InitializeAsync (
    byte[] dataBytes,
```

```
InputFileTypes fileType,
int offset = 0,
int length = -1,
int threadCount = 0,
int pageNumber = 0,
double resolutionMultiplier = 1,
bool includeAnnotations = true,
TesseractLanguage ocrLanguage = null,
CancellationToken ocrCancellationToken = default,
IProgress< OCRProgressInfo > ocrProgress = null )
```

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from an array of bytes. The OCR step is run asynchronously, in order not to block the UI thread.

Parameters

<code>dataBytes</code>	The bytes of the document that should be opened. The array will be copied and can be safely discarded/altered after this method returns.
<code>fileType</code>	The format of the document.
<code>offset</code>	The offset in the byte array at which the document starts.
<code>length</code>	The length of the document in bytes. If this is < 0, the whole array is used.
<code>threadCount</code>	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than <code>threadCount</code> that satisfies this condition is used.
<code>pageNumber</code>	The index of the page that should be rendered. The first page has index 0.
<code>resolutionMultiplier</code>	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If <code>resolutionMultiplier</code> is 1, the resolution will match the size (in screen units) of the PDFRenderer .
<code>includeAnnotations</code>	If this is <code>true</code> , annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
<code>ocrLanguage</code>	The language to use for optical character recognition (OCR). If this is <code>null</code> , no OCR is performed.
<code>ocrCancellationToken</code>	A <code>CancellationToken</code> used to cancel the OCR operation.
<code>ocrProgress</code>	An <code>IProgress<OCRProgressInfo></code> used to report OCR progress.

Definition at line 446 of file [PDFRenderer.cs](#).

7.20.4.10 InitializeAsync() [2/4]

```
async Task MuPDFCore.MuPDFRenderer.PDFRenderer.InitializeAsync (
    MemoryStream ms,
    InputFileTypes fileType,
    int threadCount = 0,
    int pageNumber = 0,
    double resolutionMultiplier = 1,
    bool includeAnnotations = true,
    TesseractLanguage ocrLanguage = null,
    CancellationToken ocrCancellationToken = default,
    IProgress< OCRProgressInfo > ocrProgress = null )
```

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from a `MemoryStream`. The OCR step is run asynchronously, in order not to block the UI thread.

Parameters

<i>ms</i>	The MemoryStream containing the document that should be opened. This can be safely disposed after this method returns.
<i>fileType</i>	The format of the document.
<i>threadCount</i>	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than <i>threadCount</i> that satisfies this condition is used.
<i>pageNumber</i>	The index of the page that should be rendered. The first page has index 0.
<i>resolutionMultiplier</i>	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If <i>resolutionMultiplier</i> is 1, the resolution will match the size (in screen units) of the PDFRenderer .
<i>includeAnnotations</i>	If this is true, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
<i>ocrCancellationToken</i>	A CancellationToken used to cancel the OCR operation.
<i>ocrProgress</i>	An IProgress<OCRProgressInfo> used to report OCR progress.

Definition at line 381 of file [PDFRenderer.cs](#).

7.20.4.11 InitializeAsync() [3/4]

```
async Task MuPDFCore.MuPDFRenderer.PDFRenderer.InitializeAsync (
    MuPDFDocument document,
    int threadCount = 0,
    int pageNumber = 0,
    double resolutionMultiplier = 1,
    bool includeAnnotations = true,
    TesseractLanguage ocrLanguage = null,
    CancellationToken ocrCancellationToken = default,
    IProgress< OCRProgressInfo > ocrProgress = null )
```

Set up the [PDFRenderer](#) to display a page of a [MuPDFDocument](#). The OCR step is run asynchronously, in order not to block the UI thread.

Parameters

<i>document</i>	The MuPDFDocument to render.
<i>threadCount</i>	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than <i>threadCount</i> that satisfies this condition is used.
<i>pageNumber</i>	The index of the page that should be rendered. The first page has index 0.
<i>resolutionMultiplier</i>	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If <i>resolutionMultiplier</i> is 1, the resolution will match the size (in screen units) of the PDFRenderer .
<i>includeAnnotations</i>	If this is true, annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is null, no OCR is performed.
<i>ocrCancellationToken</i>	A CancellationToken used to cancel the OCR operation.
<i>ocrProgress</i>	An IProgress<OCRProgressInfo> used to report OCR progress.

Definition at line 284 of file [PDFRenderer.cs](#).

7.20.4.12 InitializeAsync() [4/4]

```
async Task MuPDFCore.MuPDFRenderer.PDFRenderer.InitializeAsync (
    string fileName,
    int threadCount = 0,
    int pageNumber = 0,
    double resolutionMultiplier = 1,
    bool includeAnnotations = true,
    TesseractLanguage ocrLanguage = null,
    CancellationToken ocrCancellationToken = default,
    IProgress<OCRProgressInfo> ocrProgress = null )
```

Set up the [PDFRenderer](#) to display a page of a document that will be loaded from disk. The OCR step is run asynchronously, in order not to block the UI thread.

Parameters

<i>fileName</i>	The path to the document that should be opened.
<i>threadCount</i>	The number of threads to use in the rendering. If this is 0, an appropriate number of threads based on the number of processors in the computer will be used. Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the biggest number smaller than <i>threadCount</i> that satisfies this condition is used.
<i>pageNumber</i>	The index of the page that should be rendered. The first page has index 0.
<i>resolutionMultiplier</i>	This value can be used to increase or decrease the resolution at which the static renderisation of the page will be produced. If <i>resolutionMultiplier</i> is 1, the resolution will match the size (in screen units) of the PDFRenderer .
<i>includeAnnotations</i>	If this is <code>true</code> , annotations (e.g. signatures) are included in the rendering. Otherwise, only the page contents are included.
<i>ocrLanguage</i>	The language to use for optical character recognition (OCR). If this is <code>null</code> , no OCR is performed.
<i>ocrCancellationToken</i>	A <code>CancellationToken</code> used to cancel the OCR operation.
<i>ocrProgress</i>	An <code>IProgress<OCRProgressInfo></code> used to report OCR progress.

Definition at line 334 of file [PDFRenderer.cs](#).

7.20.4.13 ReleaseResources()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.ReleaseResources ( )
```

Release resources held by this [PDFRenderer](#). This is not an irreversible step: using one of the Initialize overloads after calling this method will restore functionality.

Definition at line 619 of file [PDFRenderer.cs](#).

7.20.4.14 Render()

```
override void MuPDFCore.MuPDFRenderer.PDFRenderer.Render (
    DrawingContext context )
```

Draw the rendered document.

Parameters

<i>context</i>	The drawing context on which to draw.
----------------	---------------------------------------

Definition at line [1299](#) of file [PDFRenderer.cs](#).

7.20.4.15 Search()

```
int MuPDFCore.MuPDFRenderer.PDFRenderer.Search (
    Regex needle )
```

Highlights all matches of the specified Regex in the text and returns the number of matches found. Matches cannot span multiple lines.

Parameters

<i>needle</i>	The Regex to search for.
---------------	--------------------------

Returns

The number of matches that have been found.

Definition at line [766](#) of file [PDFRenderer.cs](#).

7.20.4.16 SelectAll()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.SelectAll ( )
```

Selects all the text in the document.

Definition at line [745](#) of file [PDFRenderer.cs](#).

7.20.4.17 SetDisplayAreaNow()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.SetDisplayAreaNow (
    Rect value )
```

Set the current display area to the specified *value*, skipping all transitions.

Parameters

<code>value</code>	The new display area.
--------------------	-----------------------

Definition at line 660 of file [PDFRenderer.cs](#).

7.20.4.18 ZoomStep()

```
void MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomStep (
    double count,
    Point? center = null )
```

Zoom around a point.

Parameters

<code>count</code>	Number of steps to zoom. Positive values indicate a zoom in, negative values a zoom out.
<code>center</code>	The point around which to center the zoom operation. If this is null, the center of the control is used.

Definition at line 673 of file [PDFRenderer.cs](#).

7.20.5 Member Data Documentation

7.20.5.1 BackgroundProperty

```
readonly StyledProperty<IBrush> MuPDFCore.MuPDFRenderer.PDFRenderer.BackgroundProperty =
AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(Background)) [static]
```

Defines the [Background](#) property.

Definition at line 182 of file [PDFRenderer.Properties.cs](#).

7.20.5.2 DisplayAreaProperty

```
readonly StyledProperty<Rect> MuPDFCore.MuPDFRenderer.PDFRenderer.DisplayAreaProperty = AvaloniaProperty.Register<PDFRenderer, Rect>(nameof(DisplayArea)) [static]
```

Defines the [DisplayArea](#) property.

Definition at line 128 of file [PDFRenderer.Properties.cs](#).

7.20.5.3 **HighlightBrushProperty**

```
readonly StyledProperty<IBrush> MuPDFCore.MuPDFRenderer.PDFRenderer.HighlightBrushProperty  
= AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(HighlightBrush), new SolidColorBrush<  
Brush(Color.FromArgb(96, 230, 159, 0))) [static]
```

Defines the [HighlightBrush](#) property.

Definition at line 337 of file [PDFRenderer.Properties.cs](#).

7.20.5.4 **HighlightedRegionsProperty**

```
readonly StyledProperty<IEnumerable<MuPDFStructuredTextAddressSpan>> MuPDFCore.MuPDFRenderer.<  
PDFRenderer.HighlightedRegionsProperty = AvaloniaProperty.Register<PDFRenderer, IEnumerable<MuPDFStructur  
null) [static]
```

Defines the [HighlightedRegions](#) property.

Definition at line 324 of file [PDFRenderer.Properties.cs](#).

7.20.5.5 **IsViewerInitializedProperty**

```
readonly DirectProperty<PDFRenderer, bool> MuPDFCore.MuPDFRenderer.PDFRenderer.IsViewer-<  
InitializedProperty = AvaloniaProperty.RegisterDirect<PDFRenderer, bool>(nameof(IsViewerInitialized),  
o => o.IsViewerInitialized) [static]
```

Defines the [IsViewerInitialized](#) property.

Definition at line 80 of file [PDFRenderer.Properties.cs](#).

7.20.5.6 **PageBackgroundProperty**

```
readonly StyledProperty<IBrush> MuPDFCore.MuPDFRenderer.PDFRenderer.PageBackgroundProperty =  
AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(PageBackground)) [static]
```

Defines the [PageBackground](#) property.

Definition at line 195 of file [PDFRenderer.Properties.cs](#).

7.20.5.7 PageNumberProperty

```
readonly DirectProperty<PDFRenderer, int> MuPDFCore.MuPDFRenderer.PDFRenderer.PageNumber↔
Property = AvaloniaProperty.RegisterDirect<PDFRenderer, int>(nameof(PageNumber), o => o.↔
PageNumber) [static]
```

Defines the [PageNumber](#) property.

Definition at line 56 of file [PDFRenderer.Properties.cs](#).

7.20.5.8 PageSizeProperty

```
readonly DirectProperty<PDFRenderer, Rect> MuPDFCore.MuPDFRenderer.PDFRenderer.PageSize↔
Property = AvaloniaProperty.RegisterDirect<PDFRenderer, Rect>(nameof(PageSize), o => o.PageSize) [static]
```

Defines the [PageSize](#) property.

Definition at line 104 of file [PDFRenderer.Properties.cs](#).

7.20.5.9 PointerEventHandlerTypeProperty

```
readonly StyledProperty<PointerEventHandlers> MuPDFCore.MuPDFRenderer.PDFRenderer.PointerEventHandlerTypeProperty = AvaloniaProperty.Register<PDFRenderer, PointerEventHandlers>(nameof(PointerEventHandlers.PanHighlight)) [static]
```

Defines the [PointerEventHandlersType](#) property.

Definition at line 272 of file [PDFRenderer.Properties.cs](#).

7.20.5.10 RenderThreadCountProperty

```
readonly DirectProperty<PDFRenderer, int> MuPDFCore.MuPDFRenderer.PDFRenderer.RenderThreadCountProperty = AvaloniaProperty.RegisterDirect<PDFRenderer, int>(nameof(RenderThreadCount), o => o.RenderThreadCount) [static]
```

Defines the [RenderThreadCount](#) property.

Definition at line 32 of file [PDFRenderer.Properties.cs](#).

7.20.5.11 SelectionBrushProperty

```
readonly StyledProperty<IBrush> MuPDFCore.MuPDFRenderer.PDFRenderer.SelectionBrushProperty = AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(SelectionBrush), new SolidColorBrush(Color.FromArgb(96, 86, 180, 233))) [static]
```

Defines the [SelectionBrush](#) property.

Definition at line 311 of file [PDFRenderer.Properties.cs](#).

7.20.5.12 SelectionProperty

```
readonly StyledProperty<MuPDFStructuredTextAddressSpan> MuPDFCore.MuPDFRenderer.PDFRenderer.SelectionProperty = AvaloniaProperty.Register<PDFRenderer, MuPDFStructuredTextAddressSpan>(nameof(Selection), null) [static]
```

Defines the [Selection](#) property.

Definition at line 298 of file [PDFRenderer.Properties.cs](#).

7.20.5.13 ZoomEnabledProperty

```
readonly StyledProperty<bool> MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomEnabledProperty = AvaloniaProperty.Register<PDFRenderer, bool>(nameof(ZoomEnabled), true) [static]
```

Defines the [ZoomEnabled](#) property.

Definition at line 285 of file [PDFRenderer.Properties.cs](#).

7.20.5.14 ZoomIncrementProperty

```
readonly StyledProperty<double> MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomIncrementProperty = AvaloniaProperty.Register<PDFRenderer, double>(nameof(ZoomIncrement), Math.Pow(2, 1.0 / 3.0), defaultBindingMode: Avalonia.Data.BindingMode.TwoWay) [static]
```

Defines the [ZoomIncrement](#) property.

Definition at line 160 of file [PDFRenderer.Properties.cs](#).

7.20.5.15 ZoomProperty

```
readonly DirectProperty<PDFRenderer, double> MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomProperty  
= AvaloniaProperty.RegisterDirect<PDFRenderer, double>(nameof(Zoom), o => o.Zoom, (o, v) =>  
o.Zoom = v, defaultBindingMode: Avalonia.Data.BindingMode.TwoWay) [static]
```

Defines the [Zoom](#) property.

Definition at line [208](#) of file [PDFRenderer.Properties.cs](#).

7.20.6 Property Documentation

7.20.6.1 Background

```
IBrush MuPDFCore.MuPDFRenderer.PDFRenderer.Background [get], [set]
```

The background colour of the control.

Definition at line [186](#) of file [PDFRenderer.Properties.cs](#).

7.20.6.2 DisplayArea

```
Rect MuPDFCore.MuPDFRenderer.PDFRenderer.DisplayArea [get], [set]
```

The region of the page (in page units) that is currently displayed by the current instance. This always has the same aspect ratio of the bounds of this control. When this is set, the value is sanitised so that the smallest rectangle with the correct aspect ratio containing the requested value is chosen.

Definition at line [133](#) of file [PDFRenderer.Properties.cs](#).

7.20.6.3 HighlightBrush

```
IBrush MuPDFCore.MuPDFRenderer.PDFRenderer.HighlightBrush [get], [set]
```

The colour used to highlight the [HighlightedRegions](#).

Definition at line [341](#) of file [PDFRenderer.Properties.cs](#).

7.20.6.4 **HighlightedRegions**

```
IEnumerable<MuPDFStructuredTextAddressSpan> MuPDFCore.MuPDFRenderer.PDFRenderer.HighlightedRegions [get], [set]
```

A collection of highlighted regions, e.g. as a result of a text search.

Definition at line 328 of file [PDFRenderer.Properties.cs](#).

7.20.6.5 **IsViewerInitialized**

```
bool MuPDFCore.MuPDFRenderer.PDFRenderer.IsViewerInitialized [get]
```

Whether the current instance has been initialised with a document to render or not. Read-only.

Definition at line 88 of file [PDFRenderer.Properties.cs](#).

7.20.6.6 **PageBackground**

```
IBrush MuPDFCore.MuPDFRenderer.PDFRenderer.PageBackground [get], [set]
```

The background colour to use for the page drawn by the control.

Definition at line 199 of file [PDFRenderer.Properties.cs](#).

7.20.6.7 **PageNumber**

```
int MuPDFCore.MuPDFRenderer.PDFRenderer.PageNumber [get]
```

Expose the number of the page that the current instance is rendering. Read-only.

Definition at line 64 of file [PDFRenderer.Properties.cs](#).

7.20.6.8 **PageSize**

```
Rect MuPDFCore.MuPDFRenderer.PDFRenderer.PageSize [get]
```

Expose the size of the page that is drawn by the current instance (in page units).

Definition at line 112 of file [PDFRenderer.Properties.cs](#).

7.20.6.9 PointerEventHandlersType

```
PointerEventHandlers MuPDFCore.MuPDFRenderer.PDFRenderer.PointerEventHandlersType [get], [set]
```

Whether the default handlers for pointer events (which are used for panning around the page) should be enabled. If this is false, you will have to implement your own way to pan around the document by changing the [DisplayArea](#).

Definition at line [276](#) of file [PDFRenderer.Properties.cs](#).

7.20.6.10 RenderThreadCount

```
int MuPDFCore.MuPDFRenderer.PDFRenderer.RenderThreadCount [get]
```

Expose the number of threads that the current instance is using to render the document. Read-only.

Definition at line [40](#) of file [PDFRenderer.Properties.cs](#).

7.20.6.11 Selection

```
MuPDFStructuredTextAddressSpan MuPDFCore.MuPDFRenderer.PDFRenderer.Selection [get], [set]
```

The start and end of the currently selected text.

Definition at line [302](#) of file [PDFRenderer.Properties.cs](#).

7.20.6.12 SelectionBrush

```
IBrush MuPDFCore.MuPDFRenderer.PDFRenderer.SelectionBrush [get], [set]
```

The colour used to highlight the [Selection](#).

Definition at line [315](#) of file [PDFRenderer.Properties.cs](#).

7.20.6.13 Zoom

```
double MuPDFCore.MuPDFRenderer.PDFRenderer.Zoom [get], [set]
```

The current zoom level. Setting this will change the [DisplayArea](#) appropriately, zooming around the center of the [DisplayArea](#).

Definition at line [216](#) of file [PDFRenderer.Properties.cs](#).

7.20.6.14 ZoomEnabled

```
bool MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomEnabled [get], [set]
```

Whether the default handlers for pointer wheel events (which are used for zooming in/out) should be enabled. If this is false, you will have to implement your own way to zoom by changing the [DisplayArea](#).

Definition at line 289 of file [PDFRenderer.Properties.cs](#).

7.20.6.15 ZoomIncrement

```
double MuPDFCore.MuPDFRenderer.PDFRenderer.ZoomIncrement [get], [set]
```

Determines by how much the scale will be increased/decreased by the [ZoomStep\(double, Point?\)](#) method. Set this to a value smaller than 1 to invert the zoom in/out direction.

Definition at line 164 of file [PDFRenderer.Properties.cs](#).

The documentation for this class was generated from the following files:

- MuPDFCore.MuPDFRenderer/PDFRenderer.cs
- MuPDFCore.MuPDFRenderer/PDFRenderer.Properties.cs

7.21 MuPDFCore.PointF Struct Reference

Represents a point.

Public Member Functions

- [PointF \(float x, float y\)](#)
Create a new [PointF](#) from the specified coordinates.

Public Attributes

- float [X](#)
The horizontal coordinate of the point.
- float [Y](#)
The vertical coordinate of the point.

7.21.1 Detailed Description

Represents a point.

Definition at line 566 of file [Rectangles.cs](#).

7.21.2 Constructor & Destructor Documentation

7.21.2.1 PointF()

```
MuPDFCore.PointF.PointF (
    float x,
    float y )
```

Create a new [PointF](#) from the specified coordinates.

Parameters

x	The horizontal coordinate of the point.
y	The vertical coordinate of the point.

Definition at line 583 of file [Rectangles.cs](#).

7.21.3 Member Data Documentation

7.21.3.1 X

```
float MuPDFCore.PointF.X
```

The horizontal coordinate of the point.

Definition at line 571 of file [Rectangles.cs](#).

7.21.3.2 Y

```
float MuPDFCore.PointF.Y
```

The vertical coordinate of the point.

Definition at line 576 of file [Rectangles.cs](#).

The documentation for this struct was generated from the following file:

- MuPDFCore/Rectangles.cs

7.22 MuPDFCore.Quad Struct Reference

Represents a quadrilateral (not necessarily a rectangle).

Public Member Functions

- [Quad \(PointF lowerLeft, PointF upperLeft, PointF upperRight, PointF lowerRight\)](#)
*Creates a new **Quad** from the specified points.*
- [bool Contains \(PointF point\)](#)
*Checks whether this **Quad** contains a **PointF**.*

Public Attributes

- **PointF LowerLeft**
The lower left point of the quadrilater.
- **PointF UpperLeft**
The upper left point of the quadrilater.
- **PointF UpperRight**
The upper right point of the quadrilater.
- **PointF LowerRight**
The lower right point of the quadrilater.

7.22.1 Detailed Description

Represents a quadrilater (not necessarily a rectangle).

Definition at line 593 of file [Rectangles.cs](#).

7.22.2 Constructor & Destructor Documentation

7.22.2.1 Quad()

```
MuPDFCore.Quad.Quad (
    PointF lowerLeft,
    PointF upperLeft,
    PointF upperRight,
    PointF lowerRight )
```

Creates a new [Quad](#) from the specified points.

Parameters

<i>lowerLeft</i>	The lower left point of the quadrilater.
<i>upperLeft</i>	The upper left point of the quadrilater.
<i>upperRight</i>	The upper right point of the quadrilater.
<i>lowerRight</i>	The lower right point of the quadrilater.

Definition at line 622 of file [Rectangles.cs](#).

7.22.3 Member Function Documentation

7.22.3.1 Contains()

```
bool MuPDFCore.Quad.Contains (
    PointF point )
```

Checks whether this [Quad](#) contains a [PointF](#).

Parameters

<code>point</code>	The PointF to check.
--------------------	--------------------------------------

Returns

A boolean value indicating whether this [Quad](#) contains the *point*.

Definition at line 635 of file [Rectangles.cs](#).

7.22.4 Member Data Documentation

7.22.4.1 LowerLeft

[PointF](#) MuPDFCore.Quad.LowerLeft

The lower left point of the quadrilater.

Definition at line 598 of file [Rectangles.cs](#).

7.22.4.2 LowerRight

[PointF](#) MuPDFCore.Quad.LowerRight

The lower right point of the quadrilater.

Definition at line 613 of file [Rectangles.cs](#).

7.22.4.3 UpperLeft

[PointF](#) MuPDFCore.Quad.UpperLeft

The upper left point of the quadrilater.

Definition at line 603 of file [Rectangles.cs](#).

7.22.4.4 UpperRight

`PointF MuPDFCore.Quad.UpperRight`

The upper right point of the quadrilateral.

Definition at line 608 of file [Rectangles.cs](#).

The documentation for this struct was generated from the following file:

- MuPDFCore/Rectangles.cs

7.23 MuPDFCore.Rectangle Struct Reference

Represents a rectangle.

Public Member Functions

- [Rectangle](#) (float x0, float y0, float x1, float y1)
Create a new [Rectangle](#) from the specified coordinates.
- [Rectangle](#) (double x0, double y0, double x1, double y1)
Create a new [Rectangle](#) from the specified coordinates.
- [RoundedRectangle Round](#) ()
Round the rectangle's coordinates to the closest integers.
- [RoundedRectangle Round](#) (double zoom)
Round the rectangle's coordinates to the closest integers, applying the specified zoom factor.
- [Rectangle\[\] Split](#) (int divisions)
Split the rectangle into the specified number of [Rectangles](#).
- [Rectangle Intersect](#) ([Rectangle](#) other)
Compute the intersection between this [Rectangle](#) and another one.
- bool [Contains](#) ([Rectangle](#) other)
Checks whether this [Rectangle](#) contains another [Rectangle](#).
- bool [Contains](#) ([PointF](#) point)
Checks whether this [Rectangle](#) contains a [PointF](#).
- [Quad ToQuad](#) ()
Converts the [Rectangle](#) to a [Quad](#).

Public Attributes

- float [X0](#)
The left coordinate of the rectangle.
- float [Y0](#)
The top coordinate of the rectangle.
- float [X1](#)
The right coordinate of the rectangle.
- float [Y1](#)
The bottom coordinate of the rectangle.

Properties

- float [Width](#) [get]
The width of the rectangle.
- float [Height](#) [get]
The height of the rectangle.

7.23.1 Detailed Description

Represents a rectangle.

Definition at line 326 of file [Rectangles.cs](#).

7.23.2 Constructor & Destructor Documentation

7.23.2.1 Rectangle() [1/2]

```
MuPDFCore.Rectangle.Rectangle (
    float x0,
    float y0,
    float x1,
    float y1 )
```

Create a new [Rectangle](#) from the specified coordinates.

Parameters

<code>x0</code>	The left coordinate of the rectangle.
<code>y0</code>	The top coordinate of the rectangle.
<code>x1</code>	The right coordinate of the rectangle.
<code>y1</code>	The bottom coordinate of the rectangle.

Definition at line 365 of file [Rectangles.cs](#).

7.23.2.2 Rectangle() [2/2]

```
MuPDFCore.Rectangle.Rectangle (
    double x0,
    double y0,
    double x1,
    double y1 )
```

Create a new [Rectangle](#) from the specified coordinates.

Parameters

<code>x0</code>	The left coordinate of the rectangle.
<code>y0</code>	The top coordinate of the rectangle.
<code>x1</code>	The right coordinate of the rectangle.
<code>y1</code>	The bottom coordinate of the rectangle.

Definition at line 380 of file [Rectangles.cs](#).

7.23.3 Member Function Documentation

7.23.3.1 Contains() [1/2]

```
bool MuPDFCore.Rectangle.Contains (
    PointF point )
```

Checks whether this [Rectangle](#) contains a [PointF](#).

Parameters

<code>point</code>	The PointF to check.
--------------------	--------------------------------------

Returns

A boolean value indicating whether this [Rectangle](#) contains the *point*.

Definition at line 476 of file [Rectangles.cs](#).

7.23.3.2 Contains() [2/2]

```
bool MuPDFCore.Rectangle.Contains (
    Rectangle other )
```

Checks whether this [Rectangle](#) contains another [Rectangle](#).

Parameters

<code>other</code>	The Rectangle to check.
--------------------	---

Returns

A boolean value indicating whether this [Rectangle](#) contains the *other* [Rectangle](#).

Definition at line 466 of file [Rectangles.cs](#).

7.23.3.3 Intersect()

```
Rectangle MuPDFCore.Rectangle.Intersect (
    Rectangle other )
```

Compute the intersection between this [Rectangle](#) and another one.

Parameters

<i>other</i>	The other Rectangle to intersect with this instance.
--------------	--

Returns

The intersection between the two [Rectangles](#).

Definition at line 443 of file [Rectangles.cs](#).

7.23.3.4 Round() [1/2]

```
RoundedRectangle MuPDFCore.Rectangle.Round ( )
```

Round the rectangle's coordinates to the closest integers.

Returns

A [RoundedRectangle](#) with the rounded coordinates.

Definition at line 392 of file [Rectangles.cs](#).

7.23.3.5 Round() [2/2]

```
RoundedRectangle MuPDFCore.Rectangle.Round (
    double zoom )
```

Round the rectangle's coordinates to the closest integers, applying the specified zoom factor.

Parameters

<i>zoom</i>	The zoom factor to apply.
-------------	---------------------------

Returns

A [RoundedRectangle](#) with the rounded coordinates.

Definition at line 407 of file [Rectangles.cs](#).

7.23.3.6 Split()

```
Rectangle[ ] MuPDFCore.Rectangle.Split (
    int divisions )
```

Split the rectangle into the specified number of [Rectangles](#).

Parameters

<i>divisions</i>	The number of rectangles in which the rectangle should be split. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than <i>divisions</i> that satisfies this condition is used.
------------------	--

Returns

An array of [Rectangles](#) that when positioned properly cover the same area as this object.

Definition at line 422 of file [Rectangles.cs](#).

7.23.3.7 ToQuad()

```
Quad MuPDFCore.Rectangle.ToQuad ( )
```

Converts the [Rectangle](#) to a [Quad](#).

Returns

A [Quad](#) corresponding to the current [Rectangle](#).

Definition at line 485 of file [Rectangles.cs](#).

7.23.4 Member Data Documentation

7.23.4.1 X0

```
float MuPDFCore.Rectangle.X0
```

The left coordinate of the rectangle.

Definition at line 331 of file [Rectangles.cs](#).

7.23.4.2 X1

```
float MuPDFCore.Rectangle.X1
```

The right coordinate of the rectangle.

Definition at line 341 of file [Rectangles.cs](#).

7.23.4.3 Y0

```
float MuPDFCore.Rectangle.Y0
```

The top coordinate of the rectangle.

Definition at line 336 of file [Rectangles.cs](#).

7.23.4.4 Y1

```
float MuPDFCore.Rectangle.Y1
```

The bottom coordinate of the rectangle.

Definition at line 346 of file [Rectangles.cs](#).

7.23.5 Property Documentation

7.23.5.1 Height

```
float MuPDFCore.Rectangle.Height [get]
```

The height of the rectangle.

Definition at line 356 of file [Rectangles.cs](#).

7.23.5.2 Width

```
float MuPDFCore.Rectangle.Width [get]
```

The width of the rectangle.

Definition at line 351 of file [Rectangles.cs](#).

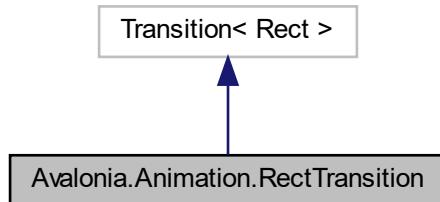
The documentation for this struct was generated from the following file:

- [MuPDFCore/Rectangles.cs](#)

7.24 Avalonia.Animation.RectTransition Class Reference

Transition class that handles AvaloniaProperty with Rect types.

Inheritance diagram for `Avalonia.Animation.RectTransition`:



Public Member Functions

- `override IObservable< Rect > DoTransition (IObservable< double > progress, Rect oldValue, Rect newValue)`

7.24.1 Detailed Description

Transition class that handles AvaloniaProperty with Rect types.

Definition at line 26 of file [RectTransition.cs](#).

7.24.2 Member Function Documentation

7.24.2.1 DoTransition()

```
override IObservable< Rect > Avalonia.Animation.RectTransition.DoTransition (
    IObservable< double > progress,
    Rect oldValue,
    Rect newValue )
```

Definition at line 29 of file [RectTransition.cs](#).

The documentation for this class was generated from the following file:

- [MuPDFCore.MuPDFRenderer/RectTransition.cs](#)

7.25 MuPDFCore.RenderProgress Class Reference

Holds a summary of the progress of the current rendering operation.

Classes

- struct [ThreadRenderProgress](#)
Holds the progress of a single thread.

Properties

- [ThreadRenderProgress\[\] ThreadRenderProgresses \[get\]](#)
Contains the progress of all the threads used in rendering the document.

7.25.1 Detailed Description

Holds a summary of the progress of the current rendering operation.

Definition at line 368 of file [MuPDF.cs](#).

7.25.2 Property Documentation

7.25.2.1 ThreadRenderProgresses

[ThreadRenderProgress \[\] MuPDFCore.RenderProgress.ThreadRenderProgresses \[get\]](#)

Contains the progress of all the threads used in rendering the document.

Definition at line 395 of file [MuPDF.cs](#).

The documentation for this class was generated from the following file:

- [MuPDFCore/MuPDF.cs](#)

7.26 MuPDFCore.RoundedRectangle Struct Reference

Represents a rectangle using only integer numbers.

Public Member Functions

- [RoundedRectangle](#) (int x0, int y0, int x1, int y1)
Create a new [RoundedRectangle](#) from the specified coordinates.
- [RoundedRectangle\[\] Split](#) (int divisions)
Split the rectangle into the specified number of [RoundedRectangles](#).

Public Attributes

- int [X0](#)
The left coordinate of the rectangle.
- int [Y0](#)
The top coordinate of the rectangle.
- int [X1](#)
The right coordinate of the rectangle.
- int [Y1](#)
The bottom coordinate of the rectangle.

Properties

- int [Width](#) [get]
The width of the rectangle.
- int [Height](#) [get]
The height of the rectangle.

7.26.1 Detailed Description

Represents a rectangle using only integer numbers.

Definition at line 494 of file [Rectangles.cs](#).

7.26.2 Constructor & Destructor Documentation

7.26.2.1 [RoundedRectangle\(\)](#)

```
MuPDFCore.RoundedRectangle.RoundedRectangle (
    int x0,
    int y0,
    int x1,
    int y1 )
```

Create a new [RoundedRectangle](#) from the specified coordinates.

Parameters

<i>x0</i>	The left coordinate of the rectangle.
<i>y0</i>	The top coordinate of the rectangle.
<i>x1</i>	The right coordinate of the rectangle.
<i>y1</i>	The bottom coordinate of the rectangle.

Definition at line 533 of file [Rectangles.cs](#).

7.26.3 Member Function Documentation

7.26.3.1 Split()

```
RoundedRectangle[] MuPDFCore.RoundedRectangle.Split (
    int divisions )
```

Split the rectangle into the specified number of [RoundedRectangles](#).

Parameters

<i>divisions</i>	The number of rectangles in which the rectangle should be split. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than <i>divisions</i> that satisfies this condition is used.
------------------	--

Returns

An array of [RoundedRectangles](#) that when positioned properly cover the same area as this object.

Definition at line 546 of file [Rectangles.cs](#).

7.26.4 Member Data Documentation

7.26.4.1 X0

```
int MuPDFCore.RoundedRectangle.X0
```

The left coordinate of the rectangle.

Definition at line 499 of file [Rectangles.cs](#).

7.26.4.2 X1

```
int MuPDFCore.RoundedRectangle.X1
```

The right coordinate of the rectangle.

Definition at line 509 of file [Rectangles.cs](#).

7.26.4.3 Y0

```
int MuPDFCore.RoundedRectangle.Y0
```

The top coordinate of the rectangle.

Definition at line 504 of file [Rectangles.cs](#).

7.26.4.4 Y1

```
int MuPDFCore.RoundedRectangle.Y1
```

The bottom coordinate of the rectangle.

Definition at line 514 of file [Rectangles.cs](#).

7.26.5 Property Documentation

7.26.5.1 Height

```
int MuPDFCore.RoundedRectangle.Height [get]
```

The height of the rectangle.

Definition at line 524 of file [Rectangles.cs](#).

7.26.5.2 Width

```
int MuPDFCore.RoundedRectangle.Width [get]
```

The width of the rectangle.

Definition at line 519 of file [Rectangles.cs](#).

The documentation for this struct was generated from the following file:

- MuPDFCore/Rectangles.cs

7.27 MuPDFCore.RoundedSize Struct Reference

Represents the size of a rectangle using only integer numbers.

Public Member Functions

- [RoundedSize](#) (int width, int height)
Create a new [RoundedSize](#) with the specified width and height.
- [RoundedRectangle\[\] Split](#) (int divisions)
Split the size into the specified number of [RoundedRectangles](#).

Public Attributes

- int [Width](#)
The width of the rectangle.
- int [Height](#)
The height of the rectangle.

7.27.1 Detailed Description

Represents the size of a rectangle using only integer numbers.

Definition at line 181 of file [Rectangles.cs](#).

7.27.2 Constructor & Destructor Documentation

7.27.2.1 RoundedSize()

```
MuPDFCore.RoundedSize.RoundedSize (
    int width,
    int height )
```

Create a new [RoundedSize](#) with the specified width and height.

Parameters

<code>width</code>	The width of the rectangle.
<code>height</code>	The height of the rectangle.

Definition at line 198 of file [Rectangles.cs](#).

7.27.3 Member Function Documentation

7.27.3.1 Split()

```
RoundedRectangle[ ] MuPDFCore.RoundedSize.Split (
    int divisions )
```

Split the size into the specified number of [RoundedRectangles](#).

Parameters

<i>divisions</i>	The number of rectangles in which the size should be split. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than <i>divisions</i> that satisfies this condition is used.
------------------	---

Returns

An array of [RoundedRectangles](#) that when positioned properly cover an area of the size of this object.

Definition at line 209 of file [Rectangles.cs](#).

7.27.4 Member Data Documentation

7.27.4.1 Height

```
int MuPDFCore.RoundedSize.Height
```

The height of the rectangle.

Definition at line 191 of file [Rectangles.cs](#).

7.27.4.2 Width

```
int MuPDFCore.RoundedSize.Width
```

The width of the rectangle.

Definition at line 186 of file [Rectangles.cs](#).

The documentation for this struct was generated from the following file:

- MuPDFCore/Rectangles.cs

7.28 MuPDFCore.Size Struct Reference

Represents the size of a rectangle.

Public Member Functions

- [Size](#) (float width, float height)
Create a new [Size](#) with the specified width and height.
- [Size](#) (double width, double height)
Create a new [Size](#) with the specified width and height.
- [Rectangle\[\] Split](#) (int divisions)
Split the size into the specified number of [Rectangles](#).

Public Attributes

- float [Width](#)
The width of the rectangle.
- float [Height](#)
The height of the rectangle.

7.28.1 Detailed Description

Represents the size of a rectangle.

Definition at line 25 of file [Rectangles.cs](#).

7.28.2 Constructor & Destructor Documentation

7.28.2.1 [Size\(\)](#) [1/2]

```
MuPDFCore.Size.Size (
    float width,
    float height )
```

Create a new [Size](#) with the specified width and height.

Parameters

<i>width</i>	The width of the rectangle.
<i>height</i>	The height of the rectangle.

Definition at line 42 of file [Rectangles.cs](#).

7.28.2.2 Size() [2/2]

```
MuPDFCore.Size.Size (
    double width,
    double height )
```

Create a new [Size](#) with the specified width and height.

Parameters

<i>width</i>	The width of the rectangle.
<i>height</i>	The height of the rectangle.

Definition at line 53 of file [Rectangles.cs](#).

7.28.3 Member Function Documentation

7.28.3.1 Split()

```
Rectangle[ ] MuPDFCore.Size.Split (
    int divisions )
```

Split the size into the specified number of [Rectangles](#).

Parameters

<i>divisions</i>	The number of rectangles in which the size should be split. This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than <i>divisions</i> that satisfies this condition is used.
------------------	---

Returns

An array of [Rectangles](#) that when positioned properly cover an area of the size of this object.

Definition at line 64 of file [Rectangles.cs](#).

7.28.4 Member Data Documentation

7.28.4.1 Height

```
float MuPDFCore.Size.Height
```

The height of the rectangle.

Definition at line 35 of file [Rectangles.cs](#).

7.28.4.2 Width

```
float MuPDFCore.Size.Width
```

The width of the rectangle.

Definition at line 30 of file [Rectangles.cs](#).

The documentation for this struct was generated from the following file:

- MuPDFCore/Rectangles.cs

7.29 MuPDFCore.TesseractLanguage Class Reference

Represents a language used by Tesseract OCR.

Public Types

- enum `Fast` {
 Afr , Amh , Ara , Asm ,
 Aze , Aze_Cyril , Bel , Ben ,
 Bod , Bos , Bre , Bul ,
 Cat , Ceb , Ces , Chi_Sim ,
 Chi_Sim_Vert , Chi_Tra , Chi_Tra_Vert , Chr ,
 Cos , Cym , Dan , Deu ,
 Div , Dzo , Ell , Eng ,
 Enm , Epo , Equ , Est ,
 Eus , Fao , Fas , Fil ,
 Fin , Fra , Frk , Frm ,
 Fry , Gla , Gle , Glg ,
 Grc , Guj , Hat , Heb ,
 Hin , Hrv , Hun , Hye ,
 Iku , Ind , Isl , Ita ,
 Ita_Old , Jav , Jpn , Jpn_Vert ,
 Kan , Kat , Kat_Old , Kaz ,
 Khm , Kir , Kmr , Kor ,
 Kor_Vert , Lao , Lat , Lav ,
 Lit , Ltz , Mal , Mar ,
 Mkd , Mlt , Mon , Mri ,
 Msa , Mya , Nep , Nld ,
 Nor , Oci , Ori , Osd ,
 Pan , Pol , Por , Pus ,
 Que , Ron , Rus , San ,
 Sin , Slk , Slv , Snd ,
 Spa , Spa_Old , Sqi , Srp ,
 Srp_Latn , Sun , Swa , Swe ,
 Syr , Tam , Tat , Tel ,
 Tgk , Tha , Tir , Ton ,
 Tur , Uig , Ukr , Urd ,
 Uzb , Uzb_Cyril , Vie , Yid ,
 Yor }

Fast integer versions of trained models. These are models for a single language.

- enum `FastScripts` {
 Arabic , Armenian , Bengali , Canadian_Aboriginal ,
 Cherokee , Cyrillic , Devanagari , Ethiopic ,
 Fraktur , Georgian , Greek , Gujarati ,
 Gurmukhi , HanS , HanS_Vert , HanT ,
 HanT_Vert , Hangul , Hangul_Vert , Hebrew ,
 Japanese , Japanese_Vert , Kannada , Khmer ,
 Lao , Latin , Malayalam , Myanmar ,
 Oriya , Sinhala , Syriac , Tamil ,
 Telugu , Thaana , Thai , Tibetan ,
 Vietnamese }

Fast integer versions of trained models. These are models for a single script supporting one or more languages.

- enum `Best` {
 Afr , Amh , Ara , Asm ,
 Aze , Aze_Cyril , Bel , Ben ,
 Bod , Bos , Bre , Bul ,
 Cat , Ceb , Ces , Chi_Sim ,
 Chi_Sim_Vert , Chi_Tra , Chi_Tra_Vert , Chr ,
 Cos , Cym , Dan , Deu ,
 Div , Dzo , Ell , Eng ,
 Enm , Epo , Est , Eus ,
 Fao , Fas , Fil , Fin ,
 Fra , Frk , Frm , Fry ,
 Gla , Gle , Glg , Grc ,
 Guj , Hat , Heb , Hin ,
 Hrv , Hun , Hye , Iku ,
 Ind , Isl , Ita , Ita_Old ,
 Jav , Jpn , Jpn_Vert , Kan ,
 Kat , Kat_Old , Kaz , Khm ,
 Kir , Kmr , Kor , Kor_Vert ,
 Lao , Lat , Lav , Lit ,
 Ltz , Mal , Mar , Mkd ,
 Mlt , Mon , Mri , Msa ,
 Mya , Nep , Nld , Nor ,
 Oci , Ori , Osd , Pan ,
 Pol , Por , Pus , Que ,
 Ron , Rus , San , Sin ,
 Slk , Slv , Snd , Spa ,
 Spa_Old , Sqi , Srp , Srp_Latn ,
 Sun , Swa , Swe , Syr ,
 Tam , Tat , Tel , Tgk ,
 Tha , Tir , Ton , Tur ,
 Uig , Ukr , Urd , Uzb ,
 Uzb_Cyril , Vie , Yid , Yor }

Best (most accurate) trained models. These are models for a single language.

- enum `BestScripts` {
 Arabic , Armenian , Bengali , Canadian_Aboriginal ,
 Cherokee , Cyrillic , Devanagari , Ethiopic ,
 Fraktur , Georgian , Greek , Gujarati ,
 Gurmukhi , HanS , HanS_Vert , HanT ,
 HanT_Vert , Hangul , Hangul_Vert , Hebrew ,
 Japanese , Japanese_Vert , Kannada , Khmer ,
 Lao , Latin , Malayalam , Myanmar ,
 Oriya , Sinhala , Syriac , Tamil ,
 Telugu , Thaana , Thai , Tibetan ,
 Vietnamese }

Best (most accurate) trained models. These are models for a single script supporting one or more languages.

Public Member Functions

- [TesseractLanguage \(string prefix, string language\)](#)
Create a new `TesseractLanguage` object using the provided prefix and language name, without processing them in any way.
- [TesseractLanguage \(string fileName\)](#)
Create a new `TesseractLanguage` object using the specified trained model data file.
- [TesseractLanguage \(Fast language, bool useAnyCached=false\)](#)
Create a new `TesseractLanguage` object using a fast integer version of a trained model for the specified language. The language file is downloaded from the `tesseract-ocr/tessdata_fast` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.
- [TesseractLanguage \(Best language, bool useAnyCached=false\)](#)
Create a new `TesseractLanguage` object using the best (most accurate) version of the trained model for the specified language. The language file is downloaded from the `tesseract-ocr/tessdata_best` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.
- [TesseractLanguage \(FastScripts script, bool useAnyCached=false\)](#)
Create a new `TesseractLanguage` object using a fast integer version of a trained model for the specified script. The language file is downloaded from the `tesseract-ocr/tessdata_fast` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.
- [TesseractLanguage \(BestScripts script, bool useAnyCached=false\)](#)
Create a new `TesseractLanguage` object using the best (most accurate) version of the trained model for the specified script. The language file is downloaded from the `tesseract-ocr/tessdata_best` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Properties

- string [Prefix \[get\]](#)
The name of the folder where the language file is located.
- string [Language \[get\]](#)
The name of the language. The Tesseract library will assume that the trained language data file can be found at `Prefix/Language.traineddata`.

7.29.1 Detailed Description

Represents a language used by Tesseract OCR.

Definition at line 13 of file [TesseractLanguage.cs](#).

7.29.2 Member Enumeration Documentation

7.29.2.1 Best

```
enum MuPDFCore.TesseractLanguage.Best
```

Best (most accurate) trained models. These are models for a single language.

Enumerator

Afr	The Afrikaans language.
Amh	The Amharic language.
Ara	The Arabic language.
Asm	The Assamese language.
Aze	The Azerbaijani language.
Aze_Cyrillic	The Azerbaijani language (Cyrillic).
Bel	The Belarusian language.
Ben	The Bengali language.
Bod	The Tibetan language.
Bos	The Bosnian language.
Bre	The Breton language.
Bul	The Bulgarian language.
Cat	The Catalan/Valencian language.
Ceb	The Cebuano language.
Ces	The Czech language.
Chi_Sim	The Chinese (Simplified) language.
Chi_Sim_Vert	The Chinese (Simplified) language (vertical).
Chi_Tra	The Chinese (Traditional) language.
Chi_Tra_Vert	The Chinese (Traditional) language (vertical).
Chr	The Cherokee language.
Cos	The Corsican language.
Cym	The Welsh language.
Dan	The Danish language.
Deu	The German language.
Div	The Divehi/Dhivehi/Maldivian language.
Dzo	The Dzongkha language.
Ell	The Greek, Modern (1453-) language.
Eng	The English language.
Enm	The English, Middle (1100-1500) language.
Epo	The Esperanto language.
Est	The Estonian language.
Eus	The Basque language.
Fao	The Faroese language.
Fas	The Persian language.
Fil	The Filipino/Pilipino language.
Fin	The Finnish language.
Fra	The French language.
Frk	The German - Fraktur language.
Frm	The French, Middle (ca.1400-1600) language.
Fry	The Western Frisian language.
Gla	The Gaelic/Scottish Gaelic language.
Gle	The Irish language.
Glg	The Galician language.
Grc	The Greek, Ancient (to 1453) language.
Guj	The Gujarati language.
Hat	The Haitian/Haitian Creole language.

Enumerator

Heb	The Hebrew language.
Hin	The Hindi language.
Hrv	The Croatian language.
Hun	The Hungarian language.
Hye	The Armenian language.
Iku	The Inuktitut language.
Ind	The Indonesian language.
Isl	The Icelandic language.
Ita	The Italian language.
Ita_Old	The Italian language (old).
Jav	The Javanese language.
Jpn	The Japanese language.
Jpn_Vert	The Japanese language (vertical).
Kan	The Kannada language.
Kat	The Georgian language.
Kat_Old	The Georgian language (old).
Kaz	The Kazakh language.
Khm	The Central Khmer language.
Kir	The Kirghiz/Kyrgyz language.
Kmr	The Northern Kurdish language.
Kor	The Korean language.
Kor_Vert	The Korean language (vertical).
Lao	The Lao language.
Lat	The Latin language.
Lav	The Latvian language.
Lit	The Lithuanian language.
Ltz	The Luxembourgish/Letzeburgesch language.
Mal	The Malayalam language.
Mar	The Marathi language.
Mkd	The Macedonian language.
Mlt	The Maltese language.
Mon	The Mongolian language.
Mri	The Maori language.
Msa	The Malay language.
Mya	The Burmese language.
Nep	The Nepali language.
Nld	The Dutch/Flemish language.
Nor	The Norwegian language.
Oci	The Occitan (post 1500) language.
Ori	The Oriya language.
Osd	The Orientation and script detection module.
Pan	The Panjabi/Punjabi language.
Pol	The Polish language.
Por	The Portuguese language.
Pus	The Pushto/Pashto language.
Que	The Quechua language.

Enumerator

Ron	The Romanian/Moldavian/Moldovan language.
Rus	The Russian language.
San	The Sanskrit language.
Sin	The Sinhala/Sinhalese language.
Slk	The Slovak language.
Slv	The Slovenian language.
Snd	The Sindhi language.
Spa	The Spanish/Castilian language.
Spa_Old	The Spanish/Castilian language (old).
Sqi	The Albanian language.
Srp	The Serbian language.
Srp_Latn	The Serbian language (Latin).
Sun	The Sundanese language.
Swahili	The Swahili language.
Swe	The Swedish language.
Syr	The Syriac language.
Tam	The Tamil language.
Tat	The Tatar language.
Tel	The Telugu language.
Tgk	The Tajik language.
Tha	The Thai language.
Tir	The Tigrinya language.
Ton	The Tonga (Tonga Islands) language.
Tur	The Turkish language.
Uig	The Uighur/Uyghur language.
Ukr	The Ukrainian language.
Urd	The Urdu language.
Uzb	The Uzbek language.
Uzb_Cyrillic	The Uzbek language (Cyrillic).
Vie	The Vietnamese language.
Yid	The Yiddish language.
Yor	The Yoruba language.

Definition at line 690 of file [TesseractLanguage.cs](#).

7.29.2.2 BestScripts

```
enum MuPDFCore.TesseractLanguage.BestScripts
```

Best (most accurate) trained models. These are models for a single script supporting one or more languages.

Enumerator

Arabic	The Arabic script.
Armenian	The Armenian script.

Enumerator

Bengali	The Bengali script.
Canadian_Aboriginal	The Canadian Aboriginal script.
Cherokee	The Cherokee script.
Cyrillic	The Cyrillic script.
Devanagari	The Devanagari script.
Ethiopic	The Ethiopic script.
Fraktur	The Fraktur script.
Georgian	The Georgian script.
Greek	The Greek script.
Gujarati	The Gujarati script.
Gurmukhi	The Gurmukhi script.
HanS	The Han (Simplified) script.
HanS_Vert	The Han (Simplified) script. (vertical)
HanT	The Han (Traditional) script.
HanT_Vert	The Han (Traditional) script. (vertical)
Hangul	The Hangul script.
Hangul_Vert	The Hangul script. (vertical)
Hebrew	The Hebrew script.
Japanese	The Japanese script.
Japanese_Vert	The Japanese script. (vertical)
Kannada	The Kannada script.
Khmer	The Khmer script.
Lao	The Lao script.
Latin	The Latin script.
Malayalam	The Malayalam script.
Myanmar	The Myanmar script.
Oriya	The Oriya script.
Sinhala	The Sinhala script.
Syriac	The Syriac script.
Tamil	The Tamil script.
Telugu	The Telugu script.
Thaana	The Thaana script.
Thai	The Thai script.
Tibetan	The Tibetan script.
Vietnamese	The Vietnamese script.

Definition at line 1193 of file [TesseractLanguage.cs](#).

7.29.2.3 Fast

```
enum MuPDFCore.TesseractLanguage.Fast
```

Fast integer versions of trained models. These are models for a single language.

Enumerator

Afr	The Afrikaans language.
Amh	The Amharic language.
Ara	The Arabic language.
Asm	The Assamese language.
Aze	The Azerbaijani language.
Aze_Cyrillic	The Azerbaijani language (Cyrillic).
Bel	The Belarusian language.
Ben	The Bengali language.
Bod	The Tibetan language.
Bos	The Bosnian language.
Bre	The Breton language.
Bul	The Bulgarian language.
Cat	The Catalan/Valencian language.
Ceb	The Cebuano language.
Ces	The Czech language.
Chi_Sim	The Chinese (Simplified) language.
Chi_Sim_Vert	The Chinese (Simplified) language (vertical).
Chi_Tra	The Chinese (Traditional) language.
Chi_Tra_Vert	The Chinese (Traditional) language (vertical).
Chr	The Cherokee language.
Cos	The Corsican language.
Cym	The Welsh language.
Dan	The Danish language.
Deu	The German language.
Div	The Divehi/Dhivehi/Maldivian language.
Dzo	The Dzongkha language.
Ell	The Greek, Modern (1453-) language.
Eng	The English language.
Enm	The English, Middle (1100-1500) language.
Epo	The Esperanto language.
Equ	A language for equations.
Est	The Estonian language.
Eus	The Basque language.
Fao	The Faroese language.
Fas	The Persian language.
Fil	The Filipino/Pilipino language.
Fin	The Finnish language.
Fra	The French language.
Frk	The German - Fraktur language.
Frm	The French, Middle (ca.1400-1600) language.
Fry	The Western Frisian language.
Gla	The Gaelic/Scottish Gaelic language.
Gle	The Irish language.
Glg	The Galician language.
Grc	The Greek, Ancient (to 1453) language.
Guj	The Gujarati language.

Enumerator

Hat	The Haitian/Haitian Creole language.
Heb	The Hebrew language.
Hin	The Hindi language.
Hrv	The Croatian language.
Hun	The Hungarian language.
Hye	The Armenian language.
Iku	The Inuktitut language.
Ind	The Indonesian language.
Isl	The Icelandic language.
Ita	The Italian language.
Ita_Old	The Italian language (old).
Jav	The Javanese language.
Jpn	The Japanese language.
Jpn_Vert	The Japanese language (vertical).
Kan	The Kannada language.
Kat	The Georgian language.
Kat_Old	The Georgian language (old).
Kaz	The Kazakh language.
Khm	The Central Khmer language.
Kir	The Kirghiz/Kyrgyz language.
Kmr	The Northern Kurdish language.
Kor	The Korean language.
Kor_Vert	The Korean language (vertical).
Lao	The Lao language.
Lat	The Latin language.
Lav	The Latvian language.
Lit	The Lithuanian language.
Ltz	The Luxembourgish/Letzeburgesch language.
Mal	The Malayalam language.
Mar	The Marathi language.
Mkd	The Macedonian language.
Mlt	The Maltese language.
Mon	The Mongolian language.
Mri	The Maori language.
Msa	The Malay language.
Mya	The Burmese language.
Nep	The Nepali language.
Nld	The Dutch/Flemish language.
Nor	The Norwegian language.
Oci	The Occitan (post 1500) language.
Ori	The Oriya language.
Osd	The Orientation and script detection module.
Pan	The Panjabi/Punjabi language.
Pol	The Polish language.
Por	The Portuguese language.
Pus	The Pushto/Pashto language.

Enumerator

Que	The Quechua language.
Ron	The Romanian/Moldavian/Moldovan language.
Rus	The Russian language.
San	The Sanskrit language.
Sin	The Sinhala/Sinhalese language.
Slk	The Slovak language.
Slv	The Slovenian language.
Snd	The Sindhi language.
Spa	The Spanish/Castilian language.
Spa_Old	The Spanish/Castilian language (old).
Sqi	The Albanian language.
Srp	The Serbian language.
Srp_Latn	The Serbian language (Latin).
Sun	The Sundanese language.
Swahili	The Swahili language.
Swe	The Swedish language.
Syr	The Syriac language.
Tam	The Tamil language.
Tat	The Tatar language.
Tel	The Telugu language.
Tgk	The Tajik language.
Tha	The Thai language.
Tir	The Tigrinya language.
Ton	The Tonga (Tonga Islands) language.
Tur	The Turkish language.
Uig	The Uighur/Uyghur language.
Ukr	The Ukrainian language.
Urd	The Urdu language.
Uzb	The Uzbek language.
Uzb_Cyril	The Uzbek language (Cyrillic).
Vie	The Vietnamese language.
Yid	The Yiddish language.
Yor	The Yoruba language.

Definition at line 28 of file [TesseractLanguage.cs](#).

7.29.2.4 FastScripts

enum [MuPDFCore.TesseractLanguage.FastScripts](#)

Fast integer versions of trained models. These are models for a single script supporting one or more languages.

Enumerator

Arabic	The Arabic script.
--------	--------------------

Enumerator

Armenian	The Armenian script.
Bengali	The Bengali script.
Canadian_Aboriginal	The Canadian Aboriginal script.
Cherokee	The Cherokee script.
Cyrillic	The Cyrillic script.
Devanagari	The Devanagari script.
Ethiopic	The Ethiopic script.
Fraktur	The Fraktur script.
Georgian	The Georgian script.
Greek	The Greek script.
Gujarati	The Gujarati script.
Gurmukhi	The Gurmukhi script.
HanS	The Han (Simplified) script.
HanS_Vert	The Han (Simplified) script. (vertical)
HanT	The Han (Traditional) script.
HanT_Vert	The Han (Traditional) script. (vertical)
Hangul	The Hangul script.
Hangul_Vert	The Hangul script. (vertical)
Hebrew	The Hebrew script.
Japanese	The Japanese script.
Japanese_Vert	The Japanese script. (vertical)
Kannada	The Kannada script.
Khmer	The Khmer script.
Lao	The Lao script.
Latin	The Latin script.
Malayalam	The Malayalam script.
Myanmar	The Myanmar script.
Oriya	The Oriya script.
Sinhala	The Sinhala script.
Syriac	The Syriac script.
Tamil	The Tamil script.
Telugu	The Telugu script.
Thaana	The Thaana script.
Thai	The Thai script.
Tibetan	The Tibetan script.
Vietnamese	The Vietnamese script.

Definition at line 535 of file [TesseractLanguage.cs](#).

7.29.3 Constructor & Destructor Documentation

7.29.3.1 TesseractLanguage() [1/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
    string prefix,
    string language )
```

Create a new [TesseractLanguage](#) object using the provided *prefix* and *language* name, without processing them in any way.

Parameters

<i>prefix</i>	The name of the folder where the language file is located. If this is <code>null</code> , the value of the environment variable <code>TESSDATA_PREFIX</code> will be used.
<i>language</i>	The name of the language. The Tesseract library will assume that the trained language data file can be found at <i>prefix</i> / <i>language</i> <code>.traineddata</code> .

Definition at line 1350 of file [TesseractLanguage.cs](#).

7.29.3.2 TesseractLanguage() [2/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
    string fileName )
```

Create a new [TesseractLanguage](#) object using the specified trained model data file.

Parameters

<i>fileName</i>	The path to the trained model data file. If the file name does not end in <code>.traineddata</code> , the file is copied to a temporary folder, and the temporary file is used by the Tesseract library.
-----------------	--

Definition at line 1360 of file [TesseractLanguage.cs](#).

7.29.3.3 TesseractLanguage() [3/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
    Fast language,
    bool useAnyCached = false )
```

Create a new [TesseractLanguage](#) object using a fast integer version of a trained model for the specified language. The language file is downloaded from the `tesseract-ocr/tessdata_fast` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Parameters

<i>language</i>	The language to use for the OCR process.
<i>useAnyCached</i>	If this is <code>true</code> , if a cached trained model file is available for the specified language, it will be used even if it is a "best (most accurate)" model. Otherwise, only cached fast integer trained models will be used.

Definition at line 1387 of file [TesseractLanguage.cs](#).

7.29.3.4 TesseractLanguage() [4/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
    Best language,
    bool useAnyCached = false )
```

Create a new [TesseractLanguage](#) object using the best (most accurate) version of the trained model for the specified language. The language file is downloaded from the `tesseract-ocr/tessdata_best` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Parameters

<code>language</code>	The language to use for the OCR process.
<code>useAnyCached</code>	If this is <code>true</code> , if a cached trained model file is available for the specified language, it will be used even if it is a "fast" model. Otherwise, only cached best (most accurate) trained models will be used.

Definition at line 1453 of file [TesseractLanguage.cs](#).

7.29.3.5 TesseractLanguage() [5/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
    FastScripts script,
    bool useAnyCached = false )
```

Create a new [TesseractLanguage](#) object using a fast integer version of a trained model for the specified script. The language file is downloaded from the `tesseract-ocr/tessdata_fast` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Parameters

<code>script</code>	The script to use for the OCR process.
<code>useAnyCached</code>	If this is <code>true</code> , if a cached trained model file is available for the specified script, it will be used even if it is a "best (most accurate)" model. Otherwise, only cached fast integer trained models will be used.

Definition at line 1519 of file [TesseractLanguage.cs](#).

7.29.3.6 TesseractLanguage() [6/6]

```
MuPDFCore.TesseractLanguage.TesseractLanguage (
    BestScripts script,
    bool useAnyCached = false )
```

Create a new `TesseractLanguage` object using the best (most accurate) version of the trained model for the specified script. The language file is downloaded from the `tesseract-ocr/tessdata_best` GitHub repository. If it has already been downloaded and cached before, the downloaded file is re-used.

Parameters

<code>script</code>	The script to use for the OCR process.
<code>useAnyCached</code>	If this is <code>true</code> , if a cached trained model file is available for the specified script, it will be used even if it is a "fast" model. Otherwise, only cached best (most accurate) trained models will be used.

Definition at line 1589 of file `TesseractLanguage.cs`.

7.29.4 Property Documentation

7.29.4.1 Language

```
string MuPDFCore.TesseractLanguage.Language [get]
```

The name of the language. The Tesseract library will assume that the trained language data file can be found at `Prefix/Language.traineddata`.

Definition at line 23 of file `TesseractLanguage.cs`.

7.29.4.2 Prefix

```
string MuPDFCore.TesseractLanguage.Prefix [get]
```

The name of the folder where the language file is located.

Definition at line 18 of file `TesseractLanguage.cs`.

The documentation for this class was generated from the following file:

- `MuPDFCore/TesseractLanguage.cs`

7.30 MuPDFCore.RenderProgress.ThreadRenderProgress Struct Reference

Holds the progress of a single thread.

Public Attributes

- int [Progress](#)
The current progress.
- long [MaxProgress](#)
The maximum progress. If this is 0, this value could not be determined (yet).

7.30.1 Detailed Description

Holds the progress of a single thread.

Definition at line 373 of file [MuPDF.cs](#).

7.30.2 Member Data Documentation

7.30.2.1 MaxProgress

```
long MuPDFCore.RenderProgress.ThreadRenderProgress.MaxProgress
```

The maximum progress. If this is 0, this value could not be determined (yet).

Definition at line 383 of file [MuPDF.cs](#).

7.30.2.2 Progress

```
int MuPDFCore.RenderProgress.ThreadRenderProgress.Progress
```

The current progress.

Definition at line 378 of file [MuPDF.cs](#).

The documentation for this struct was generated from the following file:

- MuPDFCore/MuPDF.cs

Chapter 8

File Documentation

8.1 PDFRenderer.cs

```
00001 /*
00002     MuPDFCore.MuPDFRenderer - A control to display documents in Avalonia using MuPDFCore.
00003     Copyright (C) 2020 Giorgio Bianchini
00004
00005     This program is free software: you can redistribute it and/or modify
00006     it under the terms of the GNU Affero General Public License as
00007     published by the Free Software Foundation, version 3.
00008
00009     This program is distributed in the hope that it will be useful,
00010     but WITHOUT ANY WARRANTY; without even the implied warranty of
00011     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012     GNU Affero General Public License for more details.
00013
00014     You should have received a copy of the GNU Affero General Public License
00015     along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using Avalonia;
00019 using Avalonia.Animation;
00020 using Avalonia.Controls;
00021 using Avalonia.Input;
00022 using Avalonia.Layout;
00023 using Avalonia.LogicalTree;
00024 using Avalonia.Media;
00025 using Avalonia.Media.Imaging;
00026 using Avalonia.Platform;
00027 using Avalonia.Threading;
00028 using System;
00029 using System.Collections.Generic;
00030 using System.IO;
00031 using System.Linq;
00032 using System.Runtime.InteropServices;
00033 using System.Text.RegularExpressions;
00034 using System.Threading;
00035 using System.Threading.Tasks;
00036
00037 namespace MuPDFCore.MuPDFRenderer
00038 {
00039     /// <summary>
00040     /// A control to render PDF documents (and other formats), potentially using multiple threads.
00041     /// </summary>
00042     public partial class PDFRenderer : Control
00043     {
00044         /// <summary>
00045         /// If this is true, the <see cref="Context"/> and <see cref="Document"/> will be disposed
00046         /// when this object is detached from the logical tree.
00047         private bool OwnsContextAndDocument = true;
00048
00049         /// <summary>
00050         /// The <see cref="MuPDFContext"/> using which the <see cref="Document"/> was created.
00051         /// </summary>
00052         protected MuPDFContext Context;
00053
00054         /// <summary>
00055         /// The <see cref="MuPDFDocument"/> from which the <see cref="Renderer"/> was created.
00056         /// </summary>
00057         protected MuPDFDocument Document;
```

```

00058
00059     /// <summary>
00060     /// The <see cref="MuPDFMultiThreadedPageRenderer"/> that renders the dynamic tiles.
00061     /// </summary>
00062     private MuPDFMultiThreadedPageRenderer Renderer;
00063
00064     /// <summary>
00065     /// The static renderisation of the page.
00066     /// </summary>
00067     private WriteableBitmap FixedCanvasBitmap;
00068
00069     /// <summary>
00070     /// The area covered by the <see cref="FixedCanvasBitmap"/>. It should be equal to the <see
00071     /// cref="PageSize"/>, but doesn't have to.
00072     /// </summary>
00073     private Rectangle FixedArea;
00074
00075     /// <summary>
00076     /// The position and size of the dynamic tiles.
00077     /// </summary>
00078     private RoundedRectangle[] DynamicImagesBounds;
00079
00080     /// <summary>
00081     /// The dynamic tiles.
00082     /// </summary>
00083     private WriteableBitmap[] DynamicBitmaps;
00084
00085     /// <summary>
00086     /// If this is true, the <see cref="DynamicBitmaps"/> have been rendered and can be drawn on
00087     /// screen.
00088     /// </summary>
00089     private bool AreDynamicBitmapsReady = false;
00090
00091     /// <summary>
00092     /// If this is true, the <see cref="DynamicBitmaps"/> will be rendered again immediately after
00093     /// the current rendering operation finishes.
00094     /// </summary>
00095     private bool RenderQueued = false;
00096
00097     /// <summary>
00098     /// A <see cref="Mutex"/> to synchronise rendering operations. If someone else is holding this
00099     /// mutex, you can assume that it's not safe to access the <see cref="DynamicBitmaps"/>.
00100     /// </summary>
00101     private Mutex RenderMutex;
00102
00103     /// <summary>
00104     /// A <see cref="Geometry"/> holding the icon that is displayed in the top-right corner when
00105     /// the <see cref="DynamicBitmaps"/> are not available.
00106     /// </summary>
00107     private PathGeometry RefreshingGeometry;
00108
00109     /// <summary>
00110     /// The thread that is in charge of responding to the rendering requests and either starting a
00111     /// new rendering of the <see cref="DynamicBitmaps"/>, or queueing it.
00112     /// </summary>
00113     private Thread RenderDynamicCanvasOuterThread;
00114
00115     /// <summary>
00116     /// An <see cref="EventWaitHandle"/> that signals a request for rendering to the <see
00117     /// cref="RenderDynamicCanvasOuterThread"/>.
00118     /// </summary>
00119     private readonly EventWaitHandle RenderDynamicCanvasOuterHandle = new EventWaitHandle(false,
00120     EventResetMode.ManualReset);
00121
00122     /// <summary>
00123     /// The thread that is in charge of rendering the <see cref="DynamicBitmaps"/>.
00124     /// </summary>
00125     private Thread RenderDynamicCanvasInnerThread;
00126
00127     /// <summary>
00128     /// An <see cref="EventWaitHandle"/> that signals a request for rendering to the <see
00129     /// cref="RenderDynamicCanvasInnerThread"/>.
00130     /// </summary>
00131     private readonly EventWaitHandle RenderDynamicCanvasInnerHandle = new EventWaitHandle(false,
00132     EventResetMode.ManualReset);
00133
00134     /// <summary>
00135     /// An <see cref="EventWaitHandle"/> that signals to the <see
00136     /// cref="RenderDynamicCanvasOuterThread"/> that the <see cref="RenderDynamicCanvasInnerThread"/> has
00137     /// acquired the <see cref="RenderMutex"/> and is starting rendering.
00138     /// </summary>
00139     private readonly EventWaitHandle RenderDynamicCanvasInnerStartedHandle = new
00140     EventWaitHandle(false, EventResetMode.ManualReset);
00141
00142     /// <summary>
00143     /// An <see cref="EventWaitHandle"/> that signals to the <see
00144     /// cref="RenderDynamicCanvasOuterThread"/> and the <see cref="RenderDynamicCanvasInnerThread"/> to cease

```

```
    all operation because this <see cref="PDFRenderer"/> is being detached from the logical tree.
00131    /// </summary>
00132    private readonly EventWaitHandle RendererDisposedHandle = new EventWaitHandle(false,
00133        EventResetMode.ManualReset);
00134    /// <summary>
00135    /// The current rendering resolution (in screen units) that is used by the renderer when
00136    /// rendering the <see cref="DynamicBitmaps"/>.
00137    /// </summary>
00138    private readonly int[] RenderSize = new int[2];
00139    /// <summary>
00140    /// The area on the page that will be rendered by the renderer in the <see
00141    /// cref="DynamicBitmaps"/>.
00142    /// </summary>
00143    private Rect RenderDisplayArea;
00144    /// <summary>
00145    /// A lock to prevent race conditions when multiple rendering passes are queued consecutively.
00146    /// </summary>
00147    private readonly object RenderDisplayAreaLock = new object();
00148    /// <summary>
00149    /// Whether a PointerPressed event has fired.
00150    /// </summary>
00151    private bool IsMouseDown = false;
00152    /// <summary>
00153    /// The point at which the PointerPressed event fired.
00154    /// </summary>
00155    private Point MouseDownPoint;
00156    /// <summary>
00157    /// The <see cref="DisplayArea"/> when the PointerPressed event fired.
00158    /// </summary>
00159    private Rect MouseDownDisplayArea;
00160    /// <summary>
00161    /// A structured text representation of the current page, used for selection and search
00162    /// highlight.
00163    /// </summary>
00164    protected MuPDFStructuredTextPage StructuredTextPage;
00165    /// <summary>
00166    /// A list of <see cref="Quad"/>s that cover the selected text region.
00167    /// </summary>
00168    protected List<Quad> SelectionQuads;
00169    /// <summary>
00170    /// A list of <see cref="Quad"/>s that cover the highlighted regions.
00171    /// </summary>
00172    protected List<Quad> HighlightQuads;
00173    /// <summary>
00174    /// Defines the current mouse operation.
00175    /// </summary>
00176    private enum CurrentMouseOperations
00177    {
00178        /// <summary>
00179        /// The mouse is being used to pan around the page.
00180        /// </summary>
00181        Pan,
00182        /// <summary>
00183        /// The mouse is being used to highlight text
00184        /// </summary>
00185        Highlight
00186    }
00187    /// <summary>
00188    /// The current mouse operation.
00189    /// </summary>
00190    private CurrentMouseOperations CurrentMouseOperation;
00191    /// <summary>
00192    /// Initializes a new instance of the <see cref="PDFRenderer"/> class.
00193    /// </summary>
00194    public PDFRenderer()
00195    {
00196        this.InitializeComponent();
00197
00198        this.PropertyChanged += ControlPropertyChanged;
00199        this.DetachedFromLogicalTree += ControlDetachedFromLogicalTree;
00200        this.PointerPressed += ControlPointerPressed;
00201        this.PointerReleased += ControlPointerReleased;
00202        this.PointerMoved += ControlPointerMoved;
00203        this.PointerWheelChanged += ControlPointerWheelChanged;
```

```

00213         }
00214
00215     /// <summary>
00216     /// Initializes inner components of the <see cref="PDFRenderer"/>.
00217     /// </summary>
00218     private void InitializeComponent()
00219     {
00220         PathFigure arrow1 = new PathFigure
00221         {
00222             StartPoint = new Point(16 * Math.Cos(Math.PI / 4), 16 * Math.Sin(Math.PI / 4))
00223         };
00224         arrow1.Segments.Add(new ArcSegment() { Point = new Point(-16, 0), IsLargeArc = false,
00225             RotationAngle = 0, Size = new Avalonia.Size(16, 16), SweepDirection = SweepDirection.Clockwise });
00226         arrow1.Segments.Add(new LineSegment() { Point = new Point(-7.2727, 0) });
00227         arrow1.Segments.Add(new LineSegment() { Point = new Point(-21.8181, -17.4545) });
00228         arrow1.Segments.Add(new LineSegment() { Point = new Point(-36.3636, 0) });
00229         arrow1.Segments.Add(new LineSegment() { Point = new Point(-27.6363, 0) });
00230         arrow1.Segments.Add(new ArcSegment() { Point = new Point(27.6363 * Math.Cos(Math.PI / 4),
00231             27.6363 * Math.Sin(Math.PI / 4)), IsLargeArc = false, RotationAngle = 0, Size = new
00232             Avalonia.Size(27.6363, 27.6363), SweepDirection = SweepDirection.CounterClockwise });
00233         arrow1.IsClosed = true;
00234
00235         PathFigure arrow2 = new PathFigure
00236         {
00237             StartPoint = new Point(16 * Math.Cos(5 * Math.PI / 4), 16 * Math.Sin(5 * Math.PI / 4))
00238         };
00239         arrow2.Segments.Add(new ArcSegment() { Point = new Point(16, 0), IsLargeArc = false,
00240             RotationAngle = 0, Size = new Avalonia.Size(16, 16), SweepDirection = SweepDirection.Clockwise });
00241         arrow2.Segments.Add(new LineSegment() { Point = new Point(7.2727, 0) });
00242         arrow2.Segments.Add(new LineSegment() { Point = new Point(21.8181, 17.4545) });
00243         arrow2.Segments.Add(new LineSegment() { Point = new Point(36.3636, 0) });
00244         arrow2.Segments.Add(new LineSegment() { Point = new Point(27.6363, 0) });
00245         arrow2.Segments.Add(new ArcSegment() { Point = new Point(27.6363 * Math.Cos(5 * Math.PI /
00246             4), 27.6363 * Math.Sin(5 * Math.PI / 4)), IsLargeArc = false, RotationAngle = 0, Size = new
00247             Avalonia.Size(27.6363, 27.6363), SweepDirection = SweepDirection.CounterClockwise });
00248         arrow2.IsClosed = true;
00249
00250     /// <summary>
00251     /// Set up the <see cref="PDFRenderer"/> to display a page of a <see cref="MuPDFDocument"/>.
00252     /// </summary>
00253     /// <param name="document">The <see cref="MuPDFDocument"/> to render.</param>
00254     /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an
00255     /// appropriate number of threads based on the number of processors in the computer will be used.
00256     /// Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
00257     /// biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
00258     /// used.</param>
00259     /// <param name="pageNumber">The index of the page that should be rendered. The first page has
00260     /// index 0.</param>
00261     /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00262     /// resolution at which the static renderisation of the page will be produced. If <paramref
00263     /// name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00264     /// cref="PDFRenderer"/>.</param>
00265     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00266     /// signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00267     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
00268     /// this is null, no OCR is performed.</param>
00269     public void Initialize(MuPDFDocument document, int threadCount = 0, int pageNumber = 0, double
00270     resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage ocrLanguage = null)
00271     {
00272         if (IsViewerInitialized)
00273         {
00274             ReleaseResources();
00275         }
00276
00277         OwnsContextAndDocument = false;
00278
00279         Document = document;
00280         Context = null;
00281
00282         ContinueInitialization(threadCount, pageNumber, resolutionMultiplier, includeAnnotations,
00283         ocrLanguage);
00284     }
00285
00286     /// <summary>
00287     /// Set up the <see cref="PDFRenderer"/> to display a page of a <see cref="MuPDFDocument"/>.
00288     /// The OCR step is run asynchronously, in order not to block the UI thread.
00289     /// </summary>
00290     /// <param name="document">The <see cref="MuPDFDocument"/> to render.</param>
00291     /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an
00292     /// appropriate number of threads based on the number of processors in the computer will be used.
00293     /// Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
00294     /// biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is

```

```

        used.</param>
00278     /// <param name="pageNumber">The index of the page that should be rendered. The first page has
00279     /// index 0.</param>
00280     /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00281     /// resolution at which the static renderisation of the page will be produced. If <paramref
00282     /// name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00283     /// cref="PDFRenderer"/>.</param>
00284     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00285     /// signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00286     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
00287     /// this is null, no OCR is performed.</param>
00288     /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the
00289     /// OCR operation.</param>
00290     /// <param name="ocrProgress">An <see cref="IPrgress<OCRProgressInfo>"/> used to report OCR
00291     /// progress.</param>
00292     public async Task InitializeAsync(MuPDFDocument document, int threadCount = 0, int pageNumber =
00293         0, double resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage ocrLanguage =
00294         null, CancellationToken ocrCancellationToken = default, IProgress<OCRProgressInfo> ocrProgress =
00295         null)
00296     {
00297         if (IsViewerInitialized)
00298         {
00299             ReleaseResources();
00300         }
00301
00302         OwnsContextAndDocument = false;
00303
00304         Document = document;
00305         Context = null;
00306
00307         await ContinueInitializationAsync(threadCount, pageNumber, resolutionMultiplier,
00308         includeAnnotations, ocrLanguage, ocrCancellationToken, ocrProgress);
00309     }
00310
00311     /// <summary>
00312     /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded
00313     /// from disk.
00314     /// </summary>
00315     /// <param name="fileName">The path to the document that should be opened.</param>
00316     /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an
00317     /// appropriate number of threads based on the number of processors in the computer will be used.
00318     /// Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
00319     /// biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
00320     /// used.</param>
00321     /// <param name="pageNumber">The index of the page that should be rendered. The first page has
00322     /// index 0.</param>
00323     /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00324     /// resolution at which the static renderisation of the page will be produced. If <paramref
00325     /// name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00326     /// cref="PDFRenderer"/>.</param>
00327     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00328     /// signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00329     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
00330     /// this is null, no OCR is performed.</param>
00331     public void Initialize(string fileName, int threadCount = 0, int pageNumber = 0, double
00332     resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage ocrLanguage = null)
00333     {
00334         if (IsViewerInitialized)
00335         {
00336             ReleaseResources();
00337         }
00338
00339         OwnsContextAndDocument = true;
00340
00341         Context = new MuPDFContext();
00342         Document = new MuPDFDocument(Context, fileName);
00343
00344         ContinueInitialization(threadCount, pageNumber, resolutionMultiplier, includeAnnotations,
00345         ocrLanguage);
00346     }
00347
00348     /// <summary>
00349     /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded
00350     /// from disk. The OCR step is run asynchronously, in order not to block the UI thread.
00351     /// </summary>
00352     /// <param name="fileName">The path to the document that should be opened.</param>
00353     /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an
00354     /// appropriate number of threads based on the number of processors in the computer will be used.
00355     /// Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
00356     /// biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
00357     /// used.</param>
00358     /// <param name="pageNumber">The index of the page that should be rendered. The first page has
00359     /// index 0.</param>
00360     /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00361     /// resolution at which the static renderisation of the page will be produced. If <paramref
00362     /// name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00363     /// cref="PDFRenderer"/>.</param>
```

```

00330      /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00331      signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00332      /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
00333      this is null, no OCR is performed.</param>
00334      /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the
00335      OCR operation.</param>
00336      /// <param name="ocrProgress">An <see cref="IPrinter<OCRProgressInfo>"/> used to report OCR
00337      progress.</param>
00338      public async Task InitializeAsync(string fileName, int threadCount = 0, int pageNumber = 0,
00339      double resolutionMultiplier = 1, bool includeAnnotations = true, TesseractLanguage ocrLanguage =
00340      null, CancellationToken ocrCancellationToken = default, IProgress<OCRProgressInfo> ocrProgress =
00341      null)
00342      {
00343          if (IsViewerInitialized)
00344          {
00345              ReleaseResources();
00346          }
00347          OwnsContextAndDocument = true;
00348
00349          Context = new MuPDFContext();
00350          Document = new MuPDFDocument(Context, fileName);
00351
00352          await ContinueInitializationAsync(threadCount, pageNumber, resolutionMultiplier,
00353          includeAnnotations, ocrLanguage, ocrCancellationToken, ocrProgress);
00354      }
00355
00356      /// <summary>
00357      /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded
00358      from a <see cref="MemoryStream"/>.
00359      /// </summary>
00360      /// <param name="ms">The <see cref="MemoryStream"/> containing the document that should be
00361      opened. This can be safely disposed after this method returns.</param>
00362      /// <param name="fileType">The format of the document.</param>
00363      /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an
00364      appropriate number of threads based on the number of processors in the computer will be used.
00365      Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
00366      biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
00367      used.</param>
00368      /// <param name="pageNumber">The index of the page that should be rendered. The first page has
00369      index 0.</param>
00370      /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00371      resolution at which the static renderisation of the page will be produced. If <paramref
00372      name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00373      cref="PDFRenderer"/>.</param>
00374      /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00375      signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00376      /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
00377      this is null, no OCR is performed.</param>
00378      /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the
00379      OCR operation.</param>

```

```

00380      /// <param name="ocrProgress">An <see cref="IProgress<OCRProgressInfo>" /> used to report OCR
00381      progress.</param>
00382      public async Task InitializeAsync(MemoryStream ms, InputFileTypes fileType, int threadCount =
0, int pageNumber = 0, double resolutionMultiplier = 1, bool includeAnnotations = true,
TesseractLanguage ocrLanguage = null, CancellationToken ocrCancellationToken = default,
IProgress<OCRProgressInfo> ocrProgress = null)
00383      {
00384          //Get the byte array that underlies the MemoryStream.
00385          int origin = (int)ms.Seek(0, SeekOrigin.Begin);
00386          long dataLength = ms.Length;
00387          byte[] dataBytes = ms.GetBuffer();
00388
00389          await InitializeAsync(dataBytes, fileType, origin, (int)dataLength, threadCount,
pageNumber, resolutionMultiplier, includeAnnotations, ocrLanguage, ocrCancellationToken,
ocrProgress);
00390      }
00391
00392      /// <summary>
00393      /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded
from an array of <see cref="byte"/>s.
00394      /// </summary>
00395      /// <param name="dataBytes">The bytes of the document that should be opened. The array will be
copied and can be safely discarded/altered after this method returns.</param>
00396      /// <param name="fileType">The format of the document.</param>
00397      /// <param name="offset">The offset in the byte array at which the document starts.</param>
00398      /// <param name="length">The length of the document in bytes. If this is <code>< 0</code>, the whole
array is used.</param>
00399      /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an
appropriate number of threads based on the number of processors in the computer will be used.
Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
used.</param>
00400      /// <param name="pageNumber">The index of the page that should be rendered. The first page has
index 0.</param>
00401      /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
resolution at which the static renderisation of the page will be produced. If <paramref
name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
cref="PDFRenderer"/>.</param>
00402      /// <param name="includeAnnotations">If this is <code><see langword="true" /></code>, annotations (e.g.
signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00403      /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
this is null, no OCR is performed.</param>
00404      public void Initialize(byte[] dataBytes, InputFileTypes fileType, int offset = 0, int length =
-1, int threadCount = 0, int pageNumber = 0, double resolutionMultiplier = 1, bool includeAnnotations
= true, TesseractLanguage ocrLanguage = null)
00405      {
00406          if (IsViewerInitialized)
00407          {
00408              ReleaseResources();
00409          }
00410
00411          if (length < 0)
00412          {
00413              length = dataBytes.Length - offset;
00414          }
00415
00416          //Copy the bytes to unmanaged memory, so that we don't depend on the original array.
00417          IntPtr pointer = Marshal.AllocHGlobal(length);
00418          Marshal.Copy(dataBytes, offset, pointer, length);
00419
00420          //Wrap the pointer into a disposable container.
00421          IDisposable disposer = new DisposableIntPtr(pointer);
00422
00423          OwnsContextAndDocument = true;
00424
00425          Context = new MuPDFContext();
00426
00427          //Create a new document, passing the wrapped pointer so that it can be released when the
Document is disposed.
00428          Document = new MuPDFDocument(Context, pointer, length, fileType, ref disposer);
00429
00430          ContinueInitialization(threadCount, pageNumber, resolutionMultiplier, includeAnnotations,
ocrLanguage);
00431      }
00432
00433      /// <summary>
00434      /// Set up the <see cref="PDFRenderer"/> to display a page of a document that will be loaded
from an array of <see cref="byte"/>s. The OCR step is run asynchronously, in order not to block the
UI thread.
00435      /// </summary>
00436      /// <param name="dataBytes">The bytes of the document that should be opened. The array will be
copied and can be safely discarded/altered after this method returns.</param>
00437      /// <param name="fileType">The format of the document.</param>
00438      /// <param name="offset">The offset in the byte array at which the document starts.</param>
00439      /// <param name="length">The length of the document in bytes. If this is <code>< 0</code>, the whole
array is used.</param>
00440      /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an

```

```

appropriate number of threads based on the number of processors in the computer will be used.
Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
used.</param>
00440     /// <param name="pageNumber">The index of the page that should be rendered. The first page has
00441     index 0.</param>
00442     /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00443     resolution at which the static renderisation of the page will be produced. If <paramref
00444     name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00445     cref="PDFRenderer"/>.</param>
00446     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00447     signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00448     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
00449     this is null, no OCR is performed.</param>
00450     /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the
00451     OCR operation.</param>
00452     /// <param name="ocrProgress">An <see cref="IProgress<OCRProgressInfo>"/> used to report OCR
00453     progress.</param>
00454     public async Task InitializeAsync(byte[] dataBytes, InputFileType fileType, int offset = 0,
00455     int length = -1, int threadCount = 0, int pageNumber = 0, double resolutionMultiplier = 1, bool
00456     includeAnnotations = true, TesseractLanguage ocrLanguage = null, CancellationToken
00457     ocrCancellationToken = default, IPipe<OCRProgressInfo> ocrProgress = null)
00458     {
00459         if (IsViewerInitialized)
00460         {
00461             ReleaseResources();
00462         }
00463
00464         if (length < 0)
00465         {
00466             length = dataBytes.Length - offset;
00467         }
00468
00469         //Copy the bytes to unmanaged memory, so that we don't depend on the original array.
00470         IntPtr pointer = Marshal.AllocHGlobal(length);
00471         Marshal.Copy(dataBytes, offset, pointer, length);
00472
00473         //Wrap the pointer into a disposable container.
00474         IDisposable disposer = new DisposableIntPtr(pointer);
00475
00476         OwnsContextAndDocument = true;
00477
00478         Context = new MuPDFContext();
00479
00480         //Create a new document, passing the wrapped pointer so that it can be released when the
00481         Document is disposed.
00482         Document = new MuPDFDocument(Context, pointer, length, fileType, ref disposer);
00483
00484         await ContinueInitializationAsync(threadCount, pageNumber, resolutionMultiplier,
00485         includeAnnotations, ocrLanguage, ocrCancellationToken, ocrProgress);
00486     }
00487
00488     /// <summary>
00489     /// Common steps in the initialization process that will be performed regardless of how the
00490     <see cref="Document"/> was obtained.
00491     /// </summary>
00492     /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an
00493     appropriate number of threads based on the number of processors in the computer will be used.
00494     Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
00495     biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
00496     used.</param>
00497     /// <param name="pageNumber">The index of the page that should be rendered. The first page has
00498     index 0.</param>
00499     /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00500     resolution at which the static renderisation of the page will be produced. If <paramref
00501     name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00502     cref="PDFRenderer"/>.</param>
00503     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00504     signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00505     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR).
00506     If this is null, no OCR is performed.</param>
00507     private void ContinueInitialization(int threadCount, int pageNumber, double
00508     resolutionMultiplier, bool includeAnnotations, TesseractLanguage ocrLanguage = null)
00509     {
00510         //Initialise threads and locking mechanics.
00511         if (RenderMutex == null)
00512         {
00513             RenderMutex = new Mutex(false);
00514
00515             this.RenderDynamicCanvasOuterThread = new Thread(() =>
00516             {
00517                 RenderDynamicCanvasOuterAction();
00518             });
00519
00520             this.RenderDynamicCanvasInnerThread = new Thread(() =>
00521             {
00522                 RenderDynamicCanvasInnerAction();
00523             });
00524         }
00525     }

```

```

00498         });
00499
00500         RenderDynamicCanvasOuterThread.Start();
00501         RenderDynamicCanvasInnerThread.Start();
00502     }
00503
00504     //Choose an appropriate number of threads based on the number of processors in the
00505     //computer. We have an upper limit of 8 threads because more threads apparently caused reduced
00506     //performance due to the synchronisation overhead.
00507     if (threadCount <= 0)
00508     {
00509         threadCount = Math.Max(1, Math.Min(8, Environment.ProcessorCount - 2));
00510
00511     //Create the structured text representation.
00512     this.StructuredTextPage = Document.GetStructuredTextPage(pageNumber, ocrLanguage,
00513     includeAnnotations);
00514
00515     //Create the multithreaded renderer.
00516     Renderer = Document.GetMultiThreadedRenderer(pageNumber, threadCount, includeAnnotations);
00517
00518     //Set up the properties of this control.
00519     RenderThreadCount = Renderer.ThreadCount;
00520     Rectangle bounds = Document.Pages[pageNumber].Bounds;
00521     PageSize = new Rect(new Point(bounds.X0, bounds.Y0), new Point(bounds.X1, bounds.Y1));
00522     PageNumber = pageNumber;
00523
00524     //Render the static canvas (which is used when the DynamicBitmaps are not available).
00525     RenderFixedCanvas(resolutionMultiplier);
00526
00527     //Initialize the dynamic canvas.
00528     InitializeDynamicCanvas();
00529
00530     //Set initial display area to include the whole page.
00531     double widthRatio = PageSize.Width / (this.Bounds.Width * resolutionMultiplier);
00532     double heightRatio = PageSize.Height / (this.Bounds.Height * resolutionMultiplier);
00533
00534     double containingWidth = Math.Max(widthRatio, heightRatio) * this.Bounds.Width *
00535     resolutionMultiplier;
00536     double containingHeight = Math.Max(widthRatio, heightRatio) * this.Bounds.Height *
00537     resolutionMultiplier;
00538
00539     SetDisplayAreaNowInternal(new Rect(new Point(-(containingWidth - FixedArea.Width) * 0.5,
00540     -(containingHeight - FixedArea.Height) * 0.5), new Avalonia.Size(containingWidth,
00541     containingHeight)));
00542
00543     //We are ready!
00544     IsViewerInitialized = true;
00545
00546     //Queue a render of the DynamicBitmaps (on another thread).
00547     RenderDynamicCanvas();
00548 }
00549
00550     /// <summary>
00551     /// Common steps in the initialization process that will be performed regardless of how the
00552     <see cref="Document"/> was obtained. The OCR step is run asynchronously, in order not to block the UI
00553     thread.
00554     /// </summary>
00555     /// <param name="threadCount">The number of threads to use in the rendering. If this is 0, an
00556     //appropriate number of threads based on the number of processors in the computer will be used.
00557     //Otherwise, this must be factorisable using only powers of 2, 3, 5 or 7. If this is not the case, the
00558     //biggest number smaller than <paramref name="threadCount"/> that satisfies this condition is
00559     //used.</param>
00560     /// <param name="pageNumber">The index of the page that should be rendered. The first page has
00561     index 0.</param>
00562     /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00563     resolution at which the static renderisation of the page will be produced. If <paramref
00564     name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00565     cref="PDFRenderer"/>.</param>
00566     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00567     signatures) are included in the rendering. Otherwise, only the page contents are included.</param>
00568     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
00569     this is null, no OCR is performed.</param>
00570     /// <param name="ocrCancellationToken">A <see cref="CancellationToken"/> used to cancel the
00571     OCR operation.</param>
00572     /// <param name="ocrProgress">An <see cref="IProgress<OCRProgressInfo>"/> used to report OCR
00573     progress.</param>
00574     private async Task ContinueInitializationAsync(int threadCount, int pageNumber, double
00575     resolutionMultiplier, bool includeAnnotations, TesseractLanguage ocrLanguage, CancellationToken
00576     ocrCancellationToken, IProgress<OCRProgressInfo> ocrProgress)
00577     {
00578         //Initialise threads and locking mechanics.
00579         if (RenderMutex == null)
00580         {
00581             RenderMutex = new Mutex(false);
00582
00583         }
00584
00585         RenderMutex.WaitOne();
00586
00587         //Start the threads.
00588         RenderDynamicCanvasOuterThread.Start();
00589         RenderDynamicCanvasInnerThread.Start();
00590
00591         //Wait for the threads to finish.
00592         RenderMutex.WaitOne();
00593
00594         //Release the mutex.
00595         RenderMutex.ReleaseMutex();
00596
00597         //Return the result.
00598         return StructuredTextPage;
00599     }
00600
00601     public void Dispose()
00602     {
00603         if (RenderMutex != null)
00604         {
00605             RenderMutex.Close();
00606             RenderMutex.Dispose();
00607         }
00608     }
00609
00610     protected void OnPageChanged()
00611     {
00612         if (PageNumber != PageNumber)
00613         {
00614             PageNumber = PageNumber;
00615
00616             //Requeue the render.
00617             RenderDynamicCanvas();
00618         }
00619     }
00620
00621     protected void OnPageNumberChanged()
00622     {
00623         if (PageNumber != PageNumber)
00624         {
00625             PageNumber = PageNumber;
00626
00627             //Requeue the render.
00628             RenderDynamicCanvas();
00629         }
00630     }
00631
00632     protected void OnResolutionMultiplierChanged()
00633     {
00634         if (ResolutionMultiplier != ResolutionMultiplier)
00635         {
00636             ResolutionMultiplier = ResolutionMultiplier;
00637
00638             //Requeue the render.
00639             RenderDynamicCanvas();
00640         }
00641     }
00642
00643     protected void OnIncludeAnnotationsChanged()
00644     {
00645         if (IncludeAnnotations != IncludeAnnotations)
00646         {
00647             IncludeAnnotations = IncludeAnnotations;
00648
00649             //Requeue the render.
00650             RenderDynamicCanvas();
00651         }
00652     }
00653
00654     protected void OnOcrLanguageChanged()
00655     {
00656         if (OcrLanguage != OcrLanguage)
00657         {
00658             OcrLanguage = OcrLanguage;
00659
00660             //Requeue the render.
00661             RenderDynamicCanvas();
00662         }
00663     }
00664
00665     protected void OnOcrCancellationTokenChanged()
00666     {
00667         if (OcrCancellationToken != OcrCancellationToken)
00668         {
00669             OcrCancellationToken = OcrCancellationToken;
00670
00671             //Requeue the render.
00672             RenderDynamicCanvas();
00673         }
00674     }
00675
00676     protected void OnOcrProgressChanged()
00677     {
00678         if (OcrProgress != OcrProgress)
00679         {
00680             OcrProgress = OcrProgress;
00681
00682             //Requeue the render.
00683             RenderDynamicCanvas();
00684         }
00685     }
00686
00687     protected void OnPageNumberChanged()
00688     {
00689         if (PageNumber != PageNumber)
00690         {
00691             PageNumber = PageNumber;
00692
00693             //Requeue the render.
00694             RenderDynamicCanvas();
00695         }
00696     }
00697
00698     protected void OnResolutionMultiplierChanged()
00699     {
00700         if (ResolutionMultiplier != ResolutionMultiplier)
00701         {
00702             ResolutionMultiplier = ResolutionMultiplier;
00703
00704             //Requeue the render.
00705             RenderDynamicCanvas();
00706         }
00707     }
00708
00709     protected void OnIncludeAnnotationsChanged()
00710     {
00711         if (IncludeAnnotations != IncludeAnnotations)
00712         {
00713             IncludeAnnotations = IncludeAnnotations;
00714
00715             //Requeue the render.
00716             RenderDynamicCanvas();
00717         }
00718     }
00719
00720     protected void OnOcrLanguageChanged()
00721     {
00722         if (OcrLanguage != OcrLanguage)
00723         {
00724             OcrLanguage = OcrLanguage;
00725
00726             //Requeue the render.
00727             RenderDynamicCanvas();
00728         }
00729     }
00730
00731     protected void OnOcrCancellationTokenChanged()
00732     {
00733         if (OcrCancellationToken != OcrCancellationToken)
00734         {
00735             OcrCancellationToken = OcrCancellationToken;
00736
00737             //Requeue the render.
00738             RenderDynamicCanvas();
00739         }
00740     }
00741
00742     protected void OnOcrProgressChanged()
00743     {
00744         if (OcrProgress != OcrProgress)
00745         {
00746             OcrProgress = OcrProgress;
00747
00748             //Requeue the render.
00749             RenderDynamicCanvas();
00750         }
00751     }
00752
00753     protected void OnPageNumberChanged()
00754     {
00755         if (PageNumber != PageNumber)
00756         {
00757             PageNumber = PageNumber;
00758
00759             //Requeue the render.
00760             RenderDynamicCanvas();
00761         }
00762     }
00763
00764     protected void OnResolutionMultiplierChanged()
00765     {
00766         if (ResolutionMultiplier != ResolutionMultiplier)
00767         {
00768             ResolutionMultiplier = ResolutionMultiplier;
00769
00770             //Requeue the render.
00771             RenderDynamicCanvas();
00772         }
00773     }
00774
00775     protected void OnIncludeAnnotationsChanged()
00776     {
00777         if (IncludeAnnotations != IncludeAnnotations)
00778         {
00779             IncludeAnnotations = IncludeAnnotations;
00780
00781             //Requeue the render.
00782             RenderDynamicCanvas();
00783         }
00784     }
00785
00786     protected void OnOcrLanguageChanged()
00787     {
00788         if (OcrLanguage != OcrLanguage)
00789         {
00790             OcrLanguage = OcrLanguage;
00791
00792             //Requeue the render.
00793             RenderDynamicCanvas();
00794         }
00795     }
00796
00797     protected void OnOcrCancellationTokenChanged()
00798     {
00799         if (OcrCancellationToken != OcrCancellationToken)
00800         {
00801             OcrCancellationToken = OcrCancellationToken;
00802
00803             //Requeue the render.
00804             RenderDynamicCanvas();
00805         }
00806     }
00807
00808     protected void OnOcrProgressChanged()
00809     {
00810         if (OcrProgress != OcrProgress)
00811         {
00812             OcrProgress = OcrProgress;
00813
00814             //Requeue the render.
00815             RenderDynamicCanvas();
00816         }
00817     }
00818
00819     protected void OnPageNumberChanged()
00820     {
00821         if (PageNumber != PageNumber)
00822         {
00823             PageNumber = PageNumber;
00824
00825             //Requeue the render.
00826             RenderDynamicCanvas();
00827         }
00828     }
00829
00830     protected void OnResolutionMultiplierChanged()
00831     {
00832         if (ResolutionMultiplier != ResolutionMultiplier)
00833         {
00834             ResolutionMultiplier = ResolutionMultiplier;
00835
00836             //Requeue the render.
00837             RenderDynamicCanvas();
00838         }
00839     }
00840
00841     protected void OnIncludeAnnotationsChanged()
00842     {
00843         if (IncludeAnnotations != IncludeAnnotations)
00844         {
00845             IncludeAnnotations = IncludeAnnotations;
00846
00847             //Requeue the render.
00848             RenderDynamicCanvas();
00849         }
00850     }
00851
00852     protected void OnOcrLanguageChanged()
00853     {
00854         if (OcrLanguage != OcrLanguage)
00855         {
00856             OcrLanguage = OcrLanguage;
00857
00858             //Requeue the render.
00859             RenderDynamicCanvas();
00860         }
00861     }
00862
00863     protected void OnOcrCancellationTokenChanged()
00864     {
00865         if (OcrCancellationToken != OcrCancellationToken)
00866         {
00867             OcrCancellationToken = OcrCancellationToken;
00868
00869             //Requeue the render.
00870             RenderDynamicCanvas();
00871         }
00872     }
00873
00874     protected void OnOcrProgressChanged()
00875     {
00876         if (OcrProgress != OcrProgress)
00877         {
00878             OcrProgress = OcrProgress;
00879
00880             //Requeue the render.
00881             RenderDynamicCanvas();
00882         }
00883     }
00884
00885     protected void OnPageNumberChanged()
00886     {
00887         if (PageNumber != PageNumber)
00888         {
00889             PageNumber = PageNumber;
00890
00891             //Requeue the render.
00892             RenderDynamicCanvas();
00893         }
00894     }
00895
00896     protected void OnResolutionMultiplierChanged()
00897     {
00898         if (ResolutionMultiplier != ResolutionMultiplier)
00899         {
00900             ResolutionMultiplier = ResolutionMultiplier;
00901
00902             //Requeue the render.
00903             RenderDynamicCanvas();
00904         }
00905     }
00906
00907     protected void OnIncludeAnnotationsChanged()
00908     {
00909         if (IncludeAnnotations != IncludeAnnotations)
00910         {
00911             IncludeAnnotations = IncludeAnnotations;
00912
00913             //Requeue the render.
00914             RenderDynamicCanvas();
00915         }
00916     }
00917
00918     protected void OnOcrLanguageChanged()
00919     {
00920         if (OcrLanguage != OcrLanguage)
00921         {
00922             OcrLanguage = OcrLanguage;
00923
00924             //Requeue the render.
00925             RenderDynamicCanvas();
00926         }
00927     }
00928
00929     protected void OnOcrCancellationTokenChanged()
00930     {
00931         if (OcrCancellationToken != OcrCancellationToken)
00932         {
00933             OcrCancellationToken = OcrCancellationToken;
00934
00935             //Requeue the render.
00936             RenderDynamicCanvas();
00937         }
00938     }
00939
00940     protected void OnOcrProgressChanged()
00941     {
00942         if (OcrProgress != OcrProgress)
00943         {
00944             OcrProgress = OcrProgress;
00945
00946             //Requeue the render.
00947             RenderDynamicCanvas();
00948         }
00949     }
00950
00951     protected void OnPageNumberChanged()
00952     {
00953         if (PageNumber != PageNumber)
00954         {
00955             PageNumber = PageNumber;
00956
00957             //Requeue the render.
00958             RenderDynamicCanvas();
00959         }
00960     }
00961
00962     protected void OnResolutionMultiplierChanged()
00963     {
00964         if (ResolutionMultiplier != ResolutionMultiplier)
00965         {
00966             ResolutionMultiplier = ResolutionMultiplier;
00967
00968             //Requeue the render.
00969             RenderDynamicCanvas();
00970         }
00971     }
00972
00973     protected void OnIncludeAnnotationsChanged()
00974     {
00975         if (IncludeAnnotations != IncludeAnnotations)
00976         {
00977             IncludeAnnotations = IncludeAnnotations;
00978
00979             //Requeue the render.
00980             RenderDynamicCanvas();
00981         }
00982     }
00983
00984     protected void OnOcrLanguageChanged()
00985     {
00986         if (OcrLanguage != OcrLanguage)
00987         {
00988             OcrLanguage = OcrLanguage;
00989
00990             //Requeue the render.
00991             RenderDynamicCanvas();
00992         }
00993     }
00994
00995     protected void OnOcrCancellationTokenChanged()
00996     {
00997         if (OcrCancellationToken != OcrCancellationToken)
00998         {
00999             OcrCancellationToken = OcrCancellationToken;
01000
01001             //Requeue the render.
01002             RenderDynamicCanvas();
01003         }
01004     }
01005
01006     protected void OnOcrProgressChanged()
01007     {
01008         if (OcrProgress != OcrProgress)
01009         {
01010             OcrProgress = OcrProgress;
01011
01012             //Requeue the render.
01013             RenderDynamicCanvas();
01014         }
01015     }
01016
01017     protected void OnPageNumberChanged()
01018     {
01019         if (PageNumber != PageNumber)
01020         {
01021             PageNumber = PageNumber;
01022
01023             //Requeue the render.
01024             RenderDynamicCanvas();
01025         }
01026     }
01027
01028     protected void OnResolutionMultiplierChanged()
01029     {
01030         if (ResolutionMultiplier != ResolutionMultiplier)
01031         {
01032             ResolutionMultiplier = ResolutionMultiplier;
01033
01034             //Requeue the render.
01035             RenderDynamicCanvas();
01036         }
01037     }
01038
01039     protected void OnIncludeAnnotationsChanged()
01040     {
01041         if (IncludeAnnotations != IncludeAnnotations)
01042         {
01043             IncludeAnnotations = IncludeAnnotations;
01044
01045             //Requeue the render.
01046             RenderDynamicCanvas();
01047         }
01048     }
01049
01050     protected void OnOcrLanguageChanged()
01051     {
01052         if (OcrLanguage != OcrLanguage)
01053         {
01054             OcrLanguage = OcrLanguage;
01055
01056             //Requeue the render.
01057             RenderDynamicCanvas();
01058         }
01059     }
01060
01061     protected void OnOcrCancellationTokenChanged()
01062     {
01063         if (OcrCancellationToken != OcrCancellationToken)
01064         {
01065             OcrCancellationToken = OcrCancellationToken;
01066
01067             //Requeue the render.
01068             RenderDynamicCanvas();
01069         }
01070     }
01071
01072     protected void OnOcrProgressChanged()
01073     {
01074         if (OcrProgress != OcrProgress)
01075         {
01076             OcrProgress = OcrProgress;
01077
01078             //Requeue the render.
01079             RenderDynamicCanvas();
01080         }
01081     }
01082
01083     protected void OnPageNumberChanged()
01084     {
01085         if (PageNumber != PageNumber)
01086         {
01087             PageNumber = PageNumber;
01088
01089             //Requeue the render.
01090             RenderDynamicCanvas();
01091         }
01092     }
01093
01094     protected void OnResolutionMultiplierChanged()
01095     {
01096         if (ResolutionMultiplier != ResolutionMultiplier)
01097         {
01098             ResolutionMultiplier = ResolutionMultiplier;
01099
01100             //Requeue the render.
01101             RenderDynamicCanvas();
01102         }
01103     }
01104
01105     protected void OnIncludeAnnotationsChanged()
01106     {
01107         if (IncludeAnnotations != IncludeAnnotations)
01108         {
01109             IncludeAnnotations = IncludeAnnotations;
01110
01111             //Requeue the render.
01112             RenderDynamicCanvas();
01113         }
01114     }
01115
01116     protected void OnOcrLanguageChanged()
01117     {
01118         if (OcrLanguage != OcrLanguage)
01119         {
01120             OcrLanguage = OcrLanguage;
01121
01122             //Requeue the render.
01123             RenderDynamicCanvas();
01124         }
01125     }
01126
01127     protected void OnOcrCancellationTokenChanged()
01128     {
01129         if (OcrCancellationToken != OcrCancellationToken)
01130         {
01131             OcrCancellationToken = OcrCancellationToken;
01132
01133             //Requeue the render.
01134             RenderDynamicCanvas();
01135         }
01136     }
01137
01138     protected void OnOcrProgressChanged()
01139     {
01140         if (OcrProgress != OcrProgress)
01141         {
01142             OcrProgress = OcrProgress;
01143
01144             //Requeue the render.
01145             RenderDynamicCanvas();
01146         }
01147     }
01148
01149     protected void OnPageNumberChanged()
01150     {
01151         if (PageNumber != PageNumber)
01152         {
01153             PageNumber = PageNumber;
01154
01155             //Requeue the render.
01156             RenderDynamicCanvas();
01157         }
01158     }
01159
01160     protected void OnResolutionMultiplierChanged()
01161     {
01162         if (ResolutionMultiplier != ResolutionMultiplier)
01163         {
01164             ResolutionMultiplier = ResolutionMultiplier;
01165
01166             //Requeue the render.
01167             RenderDynamicCanvas();
01168         }
01169     }
01170
01171     protected void OnIncludeAnnotationsChanged()
01172     {
01173         if (IncludeAnnotations != IncludeAnnotations)
01174         {
01175             IncludeAnnotations = IncludeAnnotations;
01176
01177             //Requeue the render.
01178             RenderDynamicCanvas();
01179         }
01180     }
01181
01182     protected void OnOcrLanguageChanged()
01183     {
01184         if (OcrLanguage != OcrLanguage)
01185         {
01186             OcrLanguage = OcrLanguage;
01187
01188             //Requeue the render.
01189             RenderDynamicCanvas();
01190         }
01191     }
01192
01193     protected void OnOcrCancellationTokenChanged()
01194     {
01195         if (OcrCancellationToken != OcrCancellationToken)
01196         {
01197             OcrCancellationToken = OcrCancellationToken;
01198
01199             //Requeue the render.
01200             RenderDynamicCanvas();
01201         }
01202     }
01203
01204     protected void OnOcrProgressChanged()
01205     {
01206         if (OcrProgress != OcrProgress)
01207         {
01208             OcrProgress = OcrProgress;
01209
01210             //Requeue the render.
01211             RenderDynamicCanvas();
01212         }
01213     }
01214
01215     protected void OnPageNumberChanged()
01216     {
01217         if (PageNumber != PageNumber)
01218         {
01219             PageNumber = PageNumber;
01220
01221             //Requeue the render.
01222             RenderDynamicCanvas();
01223         }
01224     }
01225
01226     protected void OnResolutionMultiplierChanged()
01227     {
01228         if (ResolutionMultiplier != ResolutionMultiplier)
01229         {
01230             ResolutionMultiplier = ResolutionMultiplier;
01231
01232             //Requeue the render.
01233             RenderDynamicCanvas();
01234         }
01235     }
01236
01237     protected void OnIncludeAnnotationsChanged()
01238     {
01239         if (IncludeAnnotations != IncludeAnnotations)
01240         {
01241             IncludeAnnotations = IncludeAnnotations;
01242
01243             //Requeue the render.
01244             RenderDynamicCanvas();
01245         }
01246     }
01247
01248     protected void OnOcrLanguageChanged()
01249     {
01250         if (OcrLanguage != OcrLanguage)
01251         {
01252             OcrLanguage = OcrLanguage;
01253
01254             //Requeue the render.
01255             RenderDynamicCanvas();
01256         }
01257     }
01258
01259     protected void OnOcrCancellationTokenChanged()
01260     {
01261         if (OcrCancellationToken != OcrCancellationToken)
01262         {
01263             OcrCancellationToken = OcrCancellationToken;
01264
01265             //Requeue the render.
01266             RenderDynamicCanvas();
01267         }
01268     }
01269
01270     protected void OnOcrProgressChanged()
01271     {
01272         if (OcrProgress != OcrProgress)
01273         {
01274             OcrProgress = OcrProgress;
01275
01276             //Requeue the render.
01277             RenderDynamicCanvas();
01278         }
01279     }
01280
01281     protected void OnPageNumberChanged()
01282     {
01283         if (PageNumber != PageNumber)
01284         {
01285             PageNumber = PageNumber;
01286
01287             //Requeue the render.
01288             RenderDynamicCanvas();
01289         }
01290     }
01291
01292     protected void OnResolutionMultiplierChanged()
01293     {
01294         if (ResolutionMultiplier != ResolutionMultiplier)
01295         {
01296             ResolutionMultiplier = ResolutionMultiplier;
01297
01298             //Requeue the render.
01299             RenderDynamicCanvas();
01300         }
01301     }
01302
01303     protected void OnIncludeAnnotationsChanged()
01304     {
01305         if (IncludeAnnotations != IncludeAnnotations)
01306         {
01307             IncludeAnnotations = IncludeAnnotations;
01308
01309             //Requeue the render.
01310             RenderDynamicCanvas();
01311         }
01312     }
01313
01314     protected void OnOcrLanguageChanged()
01315     {
01316         if (OcrLanguage != OcrLanguage)
01317         {
01318             OcrLanguage = OcrLanguage;
01319
01320             //Requeue the render.
01321             RenderDynamicCanvas();
01322         }
01323     }
01324
01325     protected void OnOcrCancellationTokenChanged()
01326     {
01327         if (OcrCancellationToken != OcrCancellationToken)
01328         {
01329             OcrCancellationToken = OcrCancellationToken;
01330
01331             //Requeue the render.
01332             RenderDynamicCanvas();
01333         }
01334     }
01335
01336     protected void OnOcrProgressChanged()
01337     {
01338         if (OcrProgress != OcrProgress)
01339         {
01340             OcrProgress = OcrProgress;
01341
01342             //Requeue the render.
01343             RenderDynamicCanvas();
01344         }
01345     }
01346
01347     protected void OnPageNumberChanged()
01348     {
01349         if (PageNumber != PageNumber)
01350         {
01351             PageNumber = PageNumber;
01352
01353             //Requeue the render.
01354             RenderDynamicCanvas();
01355         }
01356     }
01357
01358     protected void OnResolutionMultiplierChanged()
01359     {
01360         if (ResolutionMultiplier != ResolutionMultiplier)
01361         {
01362             ResolutionMultiplier = ResolutionMultiplier;
01363
01364             //Requeue the render.
01365             RenderDynamicCanvas();
01366         }
01367     }
01368
01369     protected void OnIncludeAnnotationsChanged()
01370     {
01371         if (IncludeAnnotations != IncludeAnnotations)
01372         {
01373             IncludeAnnotations = IncludeAnnotations;
01374
01375             //Requeue the render.
01376             RenderDynamicCanvas();
01377         }
01378     }
01379
01380     protected void OnOcrLanguageChanged()
01381     {
01382         if (OcrLanguage != OcrLanguage)
01383         {
01384             OcrLanguage = OcrLanguage;
01385
01386             //Requeue the render.
01387             RenderDynamicCanvas();
01388         }
01389     }
01390
01391     protected void OnOcrCancellationTokenChanged()
01392     {
01393         if (OcrCancellationToken != OcrCancellationToken)
01394         {
01395             OcrCancellationToken = OcrCancellationToken;
01396
01397             //Requeue the render.
01398             RenderDynamicCanvas();
01399         }
01400     }
01401
01402     protected void OnOcrProgressChanged()
01403     {
01404         if (OcrProgress != OcrProgress)
01405         {
01406             OcrProgress = OcrProgress;
01407
01408             //Requeue the render.
01409             RenderDynamicCanvas();
01410         }
01411     }
01412
01413     protected void OnPageNumberChanged()
01414     {
01415         if (PageNumber != PageNumber)
01416         {
01417             PageNumber = PageNumber;
01418
01419             //Requeue the render.
01420             RenderDynamicCanvas();
01421         }
01422     }
01423
01424     protected void OnResolutionMultiplierChanged()
01425     {
01426         if (ResolutionMultiplier != ResolutionMultiplier)
01427         {
01428             ResolutionMultiplier = ResolutionMultiplier;
01429
01430             //Requeue the render.
01431             RenderDynamicCanvas();
01432         }
01433     }
01434
01435     protected void OnIncludeAnnotationsChanged()
01436     {
01437         if (IncludeAnnotations != IncludeAnnotations)
01438         {
01439             IncludeAnnotations = IncludeAnnotations;
01440
01441             //Requeue the render.
01442             RenderDynamicCanvas();
01443         }
01444     }
01445
01446     protected void OnOcrLanguageChanged()
01447     {
01448         if (OcrLanguage != OcrLanguage)
01449         {
01450             OcrLanguage = OcrLanguage;
01451
01452             //Requeue the render.
01453             RenderDynamicCanvas();
01454         }
01455     }
01456
01457     protected void OnOcrCancellationTokenChanged()
01458     {
01459         if (OcrCancellationToken != OcrCancellationToken)
01460         {
01461             OcrCancellationToken = OcrCancellationToken;
01462
01463             //Requeue the render.
01464             RenderDynamicCanvas();
01465         }
01466     }
01467
01468     protected void OnOcrProgressChanged()
01469     {
01470         if (OcrProgress != OcrProgress)
01471         {
01472             OcrProgress = OcrProgress;
01473
01474             //Requeue the render.
01475             RenderDynamicCanvas();
01476         }
01477     }
01478
01479     protected void OnPageNumberChanged()
01480     {
01481         if (PageNumber != PageNumber)
01482         {
01483             PageNumber = PageNumber;
01484
01485             //Requeue the render.
01486             RenderDynamicCanvas();
01487         }
01488     }
01489
01490     protected void OnResolutionMultiplierChanged()
01491     {
01492         if (ResolutionMultiplier != ResolutionMultiplier)
01493         {
01494             ResolutionMultiplier = ResolutionMultiplier;
01495
01496             //Requeue the render.
01497             RenderDynamicCanvas();
01498         }
01499     }
01500
01501     protected void OnIncludeAnnotationsChanged()
01502     {
01503         if (IncludeAnnotations != IncludeAnnotations)
01504         {
01505             IncludeAnnotations = IncludeAnnotations;
01506
01507             //Requeue the render.
01508             RenderDynamicCanvas();
01509         }
01510     }
01511
01512     protected void OnOcrLanguageChanged()
01513     {
01514         if (OcrLanguage != OcrLanguage)
01515         {
01516             OcrLanguage = OcrLanguage;
01517
01518             //Requeue the render.
01519             RenderDynamicCanvas();
01520         }
01521     }
01522
01523     protected void OnOcrCancellationTokenChanged()
01524     {
01525         if (OcrCancellationToken != OcrCancellationToken)
01526         {
01527             OcrCancellationToken = OcrCancellationToken;
01528
01529             //Requeue the render.
01530             RenderDynamicCanvas();
01531         }
01532     }
01533
01534     protected void OnOcrProgressChanged()
01535     {
01536         if (OcrProgress != OcrProgress)
01537         {
01538             OcrProgress = OcrProgress;
01539
01540             //Requeue the render.
01541             RenderDynamicCanvas();
01542         }
01543     }
01544
01545     protected void OnPageNumberChanged()
01546     {
01547         if (PageNumber != PageNumber)
01548         {
01549             PageNumber = PageNumber;
01550
01551             //Requeue the render.
01552             RenderDynamicCanvas();
01553         }
01554     }
01555
01556     protected void OnResolutionMultiplierChanged()
01557     {
01558         if (ResolutionMultiplier != ResolutionMultiplier)
01559         {
01560             ResolutionMultiplier = ResolutionMultiplier;
01561
01562             //Requeue the render.
01563             RenderDynamicCanvas();
01564         }
01565     }
01566
01567     protected void OnIncludeAnnotationsChanged()
01568     {
01569         if (IncludeAnnotations != IncludeAnnotations)
01570         {
01571             IncludeAnnotations = IncludeAnnotations;
01572
01573             //Requeue the render.
01574             RenderDynamicCanvas();
01575         }
01576     }
01577
01578     protected void OnOcrLanguageChanged()
01579     {
01580         if (OcrLanguage != OcrLanguage)
01581         {
01582             OcrLanguage = OcrLanguage;
01583
01584             //Requeue the render.
01585             RenderDynamicCanvas();
01586         }
01587     }
01588
01589     protected void OnOcrCancellationTokenChanged()
01590     {
01591         if (OcrCancellationToken != OcrCancellationToken)
01592         {
01593             OcrCancellationToken = OcrCancellationToken;
01594
01595             //Requeue the render.
01596             RenderDynamicCanvas();
01597         }
01598     }
01599
0
```

```

00562         this.RenderDynamicCanvasOuterThread = new Thread(() =>
00563     {
00564         RenderDynamicCanvasOuterAction();
00565     });
00566 
00567         this.RenderDynamicCanvasInnerThread = new Thread(() =>
00568     {
00569         RenderDynamicCanvasInnerAction();
00570     });
00571 
00572         RenderDynamicCanvasOuterThread.Start();
00573         RenderDynamicCanvasInnerThread.Start();
00574     }
00575 
00576     //Choose an appropriate number of threads based on the number of processors in the
00577     //computer. We have an upper limit of 8 threads because more threads apparently caused reduced
00578     //performance due to the synchronisation overhead.
00579     if (threadCount <= 0)
00580     {
00581         threadCount = Math.Max(1, Math.Min(8, Environment.ProcessorCount - 2));
00582     }
00583 
00584     //Create the structured text representation.
00585     this.StructuredTextPage = await Document.GetStructuredTextPageAsync(pageNumber,
00586     ocrLanguage, includeAnnotations, ocrCancellationToken, ocrProgress);
00587 
00588     //Create the multithreaded renderer.
00589     Renderer = Document.GetMultiThreadedRenderer(pageNumber, threadCount, includeAnnotations);
00590 
00591     //Set up the properties of this control.
00592     RenderThreadCount = Renderer.ThreadCount;
00593     Rectangle bounds = Document.Pages[pageNumber].Bounds;
00594     PageSize = new Rect(new Point(bounds.X0, bounds.Y0), new Point(bounds.X1, bounds.Y1));
00595     PageNumber = pageNumber;
00596 
00597     //Render the static canvas (which is used when the DynamicBitmaps are not available).
00598     RenderFixedCanvas(resolutionMultiplier);
00599 
00600     //Initialize the dynamic canvas.
00601     InitializeDynamicCanvas();
00602 
00603     //Set initial display area to include the whole page.
00604     double widthRatio = PageSize.Width / (this.Bounds.Width * resolutionMultiplier);
00605     double heightRatio = PageSize.Height / (this.Bounds.Height * resolutionMultiplier);
00606 
00607     double containingWidth = Math.Max(widthRatio, heightRatio) * this.Bounds.Width *
00608     resolutionMultiplier;
00609     double containingHeight = Math.Max(widthRatio, heightRatio) * this.Bounds.Height *
00610     resolutionMultiplier;
00611 
00612     SetDisplayAreaNowInternal(new Rect(new Point(-(containingWidth - FixedArea.Width) * 0.5,
00613     -(containingHeight - FixedArea.Height) * 0.5), new Avalonia.Size(containingWidth,
00614     containingHeight)));
00615 
00616     //We are ready!
00617     IsViewerInitialized = true;
00618 
00619     //Queue a render of the DynamicBitmaps (on another thread).
00620     RenderDynamicCanvas();
00621 }
00622 
00623 /// <summary>
00624 /// Release resources held by this PDFRenderer. This is not an irreversible step: using one of
00625 /// the Initialize overloads after calling this method will restore functionality.
00626 /// </summary>
00627 public void ReleaseResources()
00628 {
00629     IsViewerInitialized = false;
00630     this.Renderer?.Dispose();
00631     this.StructuredTextPage = null;
00632     this.Selection = null;
00633     this.HighlightedRegions = null;
00634 
00635     if (OwnsContextAndDocument)
00636     {
00637         this.Document?.Dispose();
00638         this.Context?.Dispose();
00639     }
00640 }
00641 
00642 /// <summary>
00643 /// Called when the PDFRenderer is removed from the logical tree (e.g. it is removed from the
00644 /// window, or the window containing it is closed). We assume that this renderer is not needed anymore.
00645 /// This is irreversible!
00646 /// </summary>
00647 private void ControlDetachedFromLogicalTree(object sender, LogicalTreeAttachmentEventArgs e)
00648 {

```

```

00639         RendererDisposedHandle.Set();
00640         ReleaseResources();
00641     }
00642
00643
00644     /// <summary>
00645     /// Set the current display area to the specified <paramref name="value"/>, skipping all
00646     /// transitions. This also skips sanity checks of the <paramref name="value"/>, since the calling methods
00647     /// will already have performed them.
00648     /// </summary>
00649     /// <param name="value">The new display area.</param>
00650     private void SetDisplayAreaNowInternal(Rect value)
00651     {
00652         Transitions prevTransitions = this.Transitions;
00653         this.Transitions = null;
00654         SetValue(DisplayAreaProperty, value);
00655         this.Transitions = prevTransitions;
00656
00657     /// <summary>
00658     /// Set the current display area to the specified <paramref name="value"/>, skipping all
00659     /// transitions.
00660     /// </summary>
00661     /// <param name="value">The new display area.</param>
00662     public void SetDisplayAreaNow(Rect value)
00663     {
00664         Transitions prevTransitions = this.Transitions;
00665         this.Transitions = null;
00666         this.DisplayArea = value;
00667         this.Transitions = prevTransitions;
00668
00669     /// <summary>
00670     /// Zoom around a point.
00671     /// </summary>
00672     /// <param name="count">Number of steps to zoom. Positive values indicate a zoom in, negative
00673     /// values a zoom out.</param>
00674     /// <param name="center">The point around which to center the zoom operation. If this is null,
00675     /// the center of the control is used.</param>
00676     public void ZoomStep(double count, Point? center = null)
00677     {
00678         if (center == null)
00679         {
00680             center = new Point(this.Bounds.Width * 0.5, this.Bounds.Height * 0.5);
00681
00682             double currZoomX = FixedArea.Width / DisplayArea.Width;
00683             double currZoomY = FixedArea.Height / DisplayArea.Height;
00684
00685             currZoomX *= Math.Pow(ZoomIncrement, count);
00686             currZoomY *= Math.Pow(ZoomIncrement, count);
00687
00688             double currWidth = FixedArea.Width / currZoomX;
00689             double currHeight = FixedArea.Height / currZoomY;
00690
00691             double deltaW = currWidth - DisplayArea.Width;
00692             double deltaH = currHeight - DisplayArea.Height;
00693
00694             SetValue(DisplayAreaProperty, new Rect(new Point(DisplayArea.X - deltaW * center.Value.X /
00695             this.Bounds.Width, DisplayArea.Y - deltaH * center.Value.Y / this.Bounds.Height), new
00696             Point(DisplayArea.Right + deltaW * (1 - center.Value.X / this.Bounds.Width), DisplayArea.Bottom +
00697             deltaH * (1 - center.Value.Y / this.Bounds.Height)));
00698
00699     /// <summary>
00700     /// Alter the display area so that the whole page fits on screen.
00701     /// </summary>
00702     public void Contain()
00703     {
00704         //This will be sanitised by the property setter.
00705         this.DisplayArea = this.PageSize;
00706
00707     /// <summary>
00708     /// Alter the display area so that the page covers the whole surface of the <see
00709     /// cref="PDFRenderer"/> (even though parts of the page may be outside it).
00710     /// </summary>
00711     public void Cover()
00712     {
00713         double widthRatio = this.PageSize.Width / (this.Bounds.Width);
00714         double heightRatio = this.PageSize.Height / (this.Bounds.Height);
00715
00716         double containingWidth = Math.Min(widthRatio, heightRatio) * this.Bounds.Width;
00717         double containingHeight = Math.Min(widthRatio, heightRatio) * this.Bounds.Height;
00718
00719         double deltaW = (containingWidth - this.PageSize.Width) * 0.5;
00720         double deltaH = (containingHeight - this.PageSize.Height) * 0.5;

```

```
00717
00718     Rect newDispArea = new Rect(new Point(this.PageSize.X - deltaW, this.PageSize.Y - deltaH),
00719     new Point(this.PageSize.Right + deltaW, this.PageSize.Bottom + deltaH));
00720
00721     //Skip sanitation.
00722     SetValue(DisplayAreaProperty, newDispArea);
00723 }
00724
00725     /// <summary>
00726     /// Get the current rendering progress.
00727     /// </summary>
00728     /// <returns>A <see cref="RenderProgress"/> object with information about the rendering
00729     /// progress of each thread.</returns>
00730     public RenderProgress GetProgress()
00731     {
00732         return Renderer.GetProgress();
00733     }
00734
00735     /// <summary>
00736     /// Get the currently selected text.
00737     /// </summary>
00738     /// <returns>The currently selected text.</returns>
00739     public string GetSelectedText()
00740     {
00741         return this.StructuredTextPage.GetText(this.Selection);
00742     }
00743
00744     /// <summary>
00745     /// Selects all the text in the document.
00746     /// </summary>
00747     public void SelectAll()
00748     {
00749         if (this.StructuredTextPage.Count > 0)
00750         {
00751             int maxBlock = this.StructuredTextPage.Count - 1;
00752             int maxLine = this.StructuredTextPage[maxBlock].Count - 1;
00753             int maxCharacter = this.StructuredTextPage[maxBlock][maxLine].Count - 1;
00754
00755             this.Selection = new MuPDFStructuredTextAddressSpan(new MuPDFStructuredTextAddress(0,
00756             0, 0), new MuPDFStructuredTextAddress(maxBlock, maxLine, maxCharacter));
00757         }
00758         else
00759         {
00760             this.Selection = null;
00761         }
00762     }
00763
00764     /// <summary>
00765     /// Highlights all matches of the specified <see cref="Regex"/> in the text and returns the
00766     /// number of matches found. Matches cannot span multiple lines.
00767     /// </summary>
00768     /// <param name="needle">The <see cref="Regex"/> to search for.</param>
00769     /// <returns>The number of matches that have been found.</returns>
00770     public int Search(Regex needle)
00771     {
00772         List<MuPDFStructuredTextAddressSpan> spans =
00773         this.StructuredTextPage.Search(needle).ToList();
00774         this.HighlightedRegions = spans;
00775         return spans.Count;
00776     }
00777
00778     /// <summary>
00779     /// Render the <see cref="FixedCanvasBitmap"/>.
00780     /// </summary>
00781     /// <param name="resolutionMultiplier">This value can be used to increase or decrease the
00782     /// resolution at which the static renderisation of the page will be produced. If <paramref
00783     /// name="resolutionMultiplier"/> is 1, the resolution will match the size (in screen units) of the <see
00784     /// cref="PDFRenderer"/>.</param>
00785     private void RenderFixedCanvas(double resolutionMultiplier)
00786     {
00787         //Take into account DPI scaling.
00788         resolutionMultiplier *= (VisualRoot as ILayoutRoot)?.LayoutScaling ?? 1;
00789
00790         double widthRatio = PageSize.Width / (this.Bounds.Width * resolutionMultiplier);
00791         double heightRatio = PageSize.Height / (this.Bounds.Height * resolutionMultiplier);
00792
00793         double zoom = 1 / Math.Min(widthRatio, heightRatio);
00794
00795         //Render the whole page
00796         Rectangle origin = new Rectangle(0, 0, PageSize.Width, PageSize.Height);
00797
00798         FixedArea = origin;
00799
00800         RoundedRectangle roundedOrigin = origin.Round(zoom);
00801
00802         RoundedSize targetSize = new RoundedSize(roundedOrigin.Width, roundedOrigin.Height);
00803         if (FixedCanvasBitmap == null)
```

```
00796         {
00797             FixedCanvasBitmap = new WriteableBitmap(new PixelSize(targetSize.Width,
00798                 targetSize.Height), new Vector(72, 72), Avalonia.Platform.PixelFormat.Rgba8888,
00799                 AlphaFormat.Unpremul);
00800         }
00801         {
00802             if (FixedCanvasBitmap.PixelSize.Width != targetSize.Width ||
00803                 FixedCanvasBitmap.PixelSize.Height != targetSize.Height)
00804             {
00805                 FixedCanvasBitmap = new WriteableBitmap(new PixelSize(targetSize.Width,
00806                     targetSize.Height), new Vector(72, 72), Avalonia.Platform.PixelFormat.Rgba8888,
00807                     AlphaFormat.Unpremul);
00808             }
00809         }
00810         //Render the page to the FixedCanvasBitmap (without marshaling).
00811         using (ILockedFramebuffer fb = FixedCanvasBitmap.Lock())
00812         {
00813             Document.Render(PageNumber, origin, zoom, PixelFormats.RGBA, fb.Address);
00814         }
00815         /// <summary>
00816         /// Set up the <see cref="DynamicBitmaps"/> array with an appropriate number of <see
00817         /// cref="WriteableBitmap"/> of the appropriate size.
00818         /// </summary>
00819         private void InitializeDynamicCanvas()
00820         {
00821             //Take into account DPI scaling.
00822             double scale = (VisualRoot as ILayoutRoot)?.LayoutScaling ?? 1;
00823             //Acquire the render mutex (we don't want anyone to touch the DynamicBitmaps while we are
00824             //resizing them!)
00825             RenderMutex.WaitOne();
00826             RoundedSize targetSize = new RoundedSize((int)Math.Ceiling(this.Bounds.Width * scale),
00827                 (int)Math.Ceiling(this.Bounds.Height * scale));
00828             //Split the target size into an appropriate number of tiles.
00829             RoundedRectangle[] splitSizes = targetSize.Split(RenderThreadCount);
00830             DynamicImagesBounds = splitSizes;
00831             if (DynamicBitmaps == null || DynamicBitmaps.Length != RenderThreadCount)
00832             {
00833                 DynamicBitmaps = new WriteableBitmap[RenderThreadCount];
00834                 for (int i = 0; i < splitSizes.Length; i++)
00835                 {
00836                     DynamicBitmaps[i] = new WriteableBitmap(new PixelSize(splitSizes[i].Width,
00837                         splitSizes[i].Height), new Vector(72, 72), Avalonia.Platform.PixelFormat.Rgba8888,
00838                         AlphaFormat.Unpremul);
00839                 }
00840             }
00841             else
00842             {
00843                 for (int i = 0; i < splitSizes.Length; i++)
00844                 {
00845                     if (DynamicBitmaps[i].PixelSize.Width != splitSizes[i].Width ||
00846                         DynamicBitmaps[i].PixelSize.Height != splitSizes[i].Height)
00847                     {
00848                         DynamicBitmaps[i] = new WriteableBitmap(new PixelSize(splitSizes[i].Width,
00849                             splitSizes[i].Height), new Vector(72, 72), Avalonia.Platform.PixelFormat.Rgba8888,
00850                             AlphaFormat.Unpremul);
00851                     }
00852                 }
00853             }
00854             //Release the render mutex.
00855             RenderMutex.ReleaseMutex();
00856         }
00857         /// <summary>
00858         /// The outer loop that is executed by the <see cref="RenderDynamicCanvasOuterThread"/>, which
00859         /// is in charge of responding to the rendering requests and either starting a new rendering of the <see
00860         /// cref="DynamicBitmaps"/>, or queueing it.
00861         /// </summary>
00862         private void RenderDynamicCanvasOuterAction()
00863         {
00864             EventWaitHandle[] handles = new EventWaitHandle[] { RenderDynamicCanvasOuterHandle,
00865                 RendererDisposedHandle };
00866             while (true)
00867             {
00868                 int result = EventWaitHandle.WaitAny(handles);
00869                 if (result == 0)
00870                 {
00871                 }
00872             }
00873         }
00874     }
00875 }
```

```

00867         //So that we don't lose consecutive requests.
00868         RenderDynamicCanvasOuterHandle.Reset();
00869
00870         //Check if the rendering is already in progress.
00871         if (RenderMutex.WaitOne(0))
00872         {
00873             //Start a new rendering
00874             AreDynamicBitmapsReady = false;
00875
00876             //This handle will be set by the inner thread once it starts rendering.
00877             RenderDynamicCanvasInnerStartedHandle.Reset();
00878
00879             //Tell the inner thread to start rendering.
00880             RenderDynamicCanvasInnerHandle.Set();
00881
00882             //Release the mutex so that the inner thread can start rendering.
00883             RenderMutex.ReleaseMutex();
00884
00885             //Wait until the inner thread has acuired the mutex and started rendering.
00886             RenderDynamicCanvasInnerStartedHandle.WaitOne();
00887         }
00888         else
00889         {
00890             if (!RenderQueued)
00891             {
00892                 //Queue another rendering pass.
00893                 RenderQueued = true;
00894
00895                 //Abort the current rendering pass.
00896                 Renderer.Abort();
00897             }
00898         }
00899     }
00900     else
00901     {
00902         //The renderer is being disposed, we need to quit!
00903         break;
00904     }
00905 }
00906 }
00907
00908 /// <summary>
00909     /// The inner loop that is executed by the <see cref="RenderDynamicCanvasInnerThread"/>, which
00910     /// renders the <see cref="DynamicBitmaps"/>.
00911     /// </summary>
00912     private void RenderDynamicCanvasInnerAction()
00913     {
00914         EventWaitHandle[] handles = new EventWaitHandle[] { RenderDynamicCanvasInnerHandle,
00915             RendererDisposedHandle };
00916
00917         bool ownsMutex = false;
00918
00919         while (true)
00920         {
00921             int result = EventWaitHandle.WaitAny(handles);
00922
00923             if (result == 0)
00924             {
00925                 //So that we don't lose consecutive requests.
00926                 RenderDynamicCanvasInnerHandle.Reset();
00927
00928                 //Acquire the mutex only if have not acquired it yet.
00929                 if (!ownsMutex)
00930                 {
00931                     RenderMutex.WaitOne();
00932                     ownsMutex = true;
00933
00934                 //Signal to the outer thread that we have acquired the mutex. Even if the outer
00935                 //thread is not waiting for this signal, it will reset it before waiting for it.
00936                 RenderDynamicCanvasInnerStartedHandle.Set();
00937
00938                 //Set up the pointers to the contents of the DynamicBitmaps
00939                 IntPtr[] destinations = new IntPtr[RenderThreadCount];
00940                 ILockedFramebuffer[] fbs = new ILockedFramebuffer[RenderThreadCount];
00941
00942                 for (int i = 0; i < RenderThreadCount; i++)
00943                 {
00944                     fbs[i] = DynamicBitmaps[i].Lock();
00945                     destinations[i] = fbs[i].Address;
00946
00947                     //Prevent race conditions.
00948                     Rectangle target;
00949                     int width;
00950                     int height;
00951                     lock (RenderDisplayAreaLock)

```

```

00951             {
00952                 target = new Rectangle(RenderDisplayArea.X, RenderDisplayArea.Y,
00953                     RenderDisplayArea.Right, RenderDisplayArea.Bottom);
00954                 width = RenderSize[0];
00955                 height = RenderSize[1];
00956             }
00957
00958             //Start the multithreaded rendering and wait until it finishes.
00959             Renderer.Render(new RoundedSize(width, height), target, destinations,
00960             PixelFormats.RGBA);
00961
00962             //Free the pointers.
00963             for (int i = 0; i < RenderThreadCount; i++)
00964             {
00965                 fbs[i].Dispose();
00966             }
00967
00968             //Check whether another rendering request has been queued.
00969             if (!RenderQueued)
00970             {
00971                 //No other rendering requests have been queued.
00972                 AreDynamicBitmapsReady = true;
00973
00974                 //This should always be true. Release the rendering mutex.
00975                 if (ownsMutex)
00976                 {
00977                     RenderMutex.ReleaseMutex();
00978                     ownsMutex = false;
00979                 }
00980
00981                 //Signal to the UI that a repaint is required.
00982                 Dispatcher.UIThread.InvokeAsync(() =>
00983                 {
00984                     this.InvalidateVisual();
00985                 });
00986             }
00987
00988             //Another rendering request has been queued, we can assume that whatever we
00989             //have rendered until now is useless (maybe because the rendering has been aborted).
00990             RenderQueued = false;
00991
00992             //Self-signal.
00993             RenderDynamicCanvasInnerHandle.Set();
00994         }
00995     }
00996
00997     //The renderer is being disposed, we need to quit!
00998     break;
00999 }
01000 }
01001
01002 /// <summary>
01003 /// Signal to the <see cref="RenderDynamicCanvasOuterThread"/> that a rendering has been
01004 requested.
01005 /// </summary>
01006 private void RenderDynamicCanvas()
01007 {
01008     //Take into account DPI scaling.
01009     double scale = (VisualRoot as ILayoutRoot).LayoutScaling;
01010
01011     //Set up rendering size
01012     lock (RenderDisplayAreaLock)
01013     {
01014         RenderSize[0] = (int)Math.Ceiling(this.Bounds.Width * scale);
01015         RenderSize[1] = (int)Math.Ceiling(this.Bounds.Height * scale);
01016         RenderDisplayArea = DisplayArea;
01017     }
01018
01019     //Send the signal.
01020     RenderDynamicCanvasOuterHandle.Set();
01021 }
01022
01023 /// <summary>
01024 /// Resizes the <see cref="DynamicBitmaps"/> when the size of the control changes and queues a
01025 repaint when the <see cref="DisplayArea"/> changes.
01026 /// </summary>
01027 /// <param name="sender"></param>
01028 /// <param name="e"></param>
01029 private void ControlPropertyChanged(object sender, AvaloniaPropertyChangedEventArgs e)
01030 {
01031     if (e.Property == UserControl.BoundsProperty)
01032     {
01033         if (IsViewerInitialized)
01034         {

```

```

01033             //Resize the display area
01034             Rect oldBounds = (Rect)e.OldValue;
01035             Rect newBounds = (Rect)e.NewValue;
01036
01037             double deltaW = (newBounds.Width - oldBounds.Width) / oldBounds.Width *
01038                 DisplayArea.Width;
01039             double deltaH = (newBounds.Height - oldBounds.Height) / oldBounds.Height *
01040                 DisplayArea.Height;
01041
01042             Rect target = new Rect(new Point(DisplayArea.X - deltaW * 0.5, DisplayArea.Y -
01043                 deltaH * 0.5), new Point(DisplayArea.Right + deltaW * 0.5, DisplayArea.Bottom + deltaH * 0.5));
01044
01045             //Resize the DynamicBitmaps
01046             InitializeDynamicCanvas();
01047
01048             //Set the new DisplayArea, skipping any animation.
01049             SetDisplayAreaNowInternal(target);
01050         }
01051     }
01052     else if (e.Property == PDFRenderer.DisplayAreaProperty)
01053     {
01054         if (IsViewerInitialized)
01055         {
01056             //Update the value of the Zoom property.
01057             ComputeZoom();
01058
01059             //Signal that a repaint is needed
01060             this.InvalidateVisual();
01061
01062         }
01063         else if (e.Property == PDFRenderer.SelectionProperty && this.StructuredTextPage != null)
01064         {
01065             //Update the selection quads to reflect the new selection
01066             this.SelectionQuads = this.StructuredTextPage.GetHighlightQuads(this.Selection,
01067                 false).ToList();
01068             this.InvalidateVisual();
01069         }
01070         else if (e.Property == PDFRenderer.HighlightedRegionsProperty && this.StructuredTextPage
01071             != null)
01072         {
01073             //Update the highlight quads to reflect the new highlighted regions
01074             this.HighlightQuads = new List<Quad>();
01075
01076             if (this.HighlightedRegions != null)
01077             {
01078                 foreach (MuPDFStructuredTextAddressSpan span in this.HighlightedRegions)
01079                 {
01080                     this.HighlightQuads.AddRange(this.StructuredTextPage.GetHighlightQuads(span,
01081                         false));
01082                 }
01083             }
01084         }
01085
01086         /// <summary>
01087         /// Default handler for the PointerPressed event (start panning).
01088         /// </summary>
01089         /// <param name="sender"></param>
01090         /// <param name="e"></param>
01091         private void ControlPointerPressed(object sender, PointerPressedEventArgs e)
01092         {
01093             if (PointerEventHandlersType == PointerEventHandlers.Pan)
01094             {
01095                 if (e.GetCurrentPoint(this).Properties.PointerUpdateKind ==
01096                     PointerUpdateKind.LeftButtonPressed)
01097                 {
01098                     IsMouseDown = true;
01099                     MouseDownPoint = eGetPosition(this);
01100                     MouseDownDisplayArea = DisplayArea;
01101                     this.Cursor = new Cursor(StandardCursorType.SizeAll);
01102                 }
01103             }
01104             else if (PointerEventHandlersType == PointerEventHandlers.Highlight)
01105             {
01106                 if (e.GetCurrentPoint(this).Properties.PointerUpdateKind ==
01107                     PointerUpdateKind.LeftButtonPressed)
01108                 {
01109                     Point point = eGetPosition(this);
01110                     IsMouseDown = true;
01111                     MouseDownPoint = point;
01112                     MouseDownDisplayArea = DisplayArea;
01113                 }
01114             }
01115         }
01116     }
01117 }

```

```

0112
0113             PointF pagePoint = new PointF((float)(point.X / this.Bounds.Width *
DisplayArea.Width + DisplayArea.Left), (float)(point.Y / this.Bounds.Height * DisplayArea.Height +
DisplayArea.Top));
0114
0115             MuPDFStructuredTextAddress? address = StructuredTextPage?.GetHitAddress(pagePoint,
false);
0116
0117             if (address != null)
0118             {
0119                 this.Selection = new MuPDFStructuredTextAddressSpan(address.Value, null);
0120             }
0121             else
0122             {
0123                 this.Selection = null;
0124             }
0125         }
0126     }
0127     else if (PointerEventHandlersType == PointerEventHandlers.PanHighlight)
0128     {
0129         Point point = e.GetPosition(this);
0130         PointF pagePoint = new PointF((float)(point.X / this.Bounds.Width * DisplayArea.Width +
DisplayArea.Left), (float)(point.Y / this.Bounds.Height * DisplayArea.Height + DisplayArea.Top));
0131         MuPDFStructuredTextAddress? address = StructuredTextPage?.GetHitAddress(pagePoint,
false);
0132
0133         if (address == null)
0134         {
0135             if (e.GetCurrentPoint(this).Properties.PointerUpdateKind ==
PointerUpdateKind.LeftButtonPressed)
0136             {
0137                 IsMouseDown = true;
0138                 MouseDownPoint = e.GetPosition(this);
0139                 MouseDownDisplayArea = DisplayArea;
0140                 this.Cursor = new Cursor(StandardCursorType.SizeAll);
0141                 CurrentMouseOperation = CurrentMouseOperations.Pan;
0142             }
0143         }
0144         else
0145         {
0146             if (e.GetCurrentPoint(this).Properties.PointerUpdateKind ==
PointerUpdateKind.LeftButtonPressed)
0147             {
0148                 IsMouseDown = true;
0149                 MouseDownPoint = point;
0150                 MouseDownDisplayArea = DisplayArea;
0151
0152                 this.Selection = new MuPDFStructuredTextAddressSpan(address.Value, null);
0153                 CurrentMouseOperation = CurrentMouseOperations.Highlight;
0154             }
0155         }
0156     }
0157 }
0158
0159 /// <summary>
0160 /// Default handler for the PointerReleased event (finish panning).
0161 /// </summary>
0162 /// <param name="sender"></param>
0163 /// <param name="e"></param>
0164 private void ControlPointerReleased(object sender, PointerReleasedEventArgs e)
0165 {
0166     if (PointerEventHandlersType == PointerEventHandlers.Pan)
0167     {
0168         if (e.InitialPressMouseButton == MouseButton.Left)
0169         {
0170             IsMouseDown = false;
0171             this.Cursor = new Cursor(StandardCursorType.Arrow);
0172         }
0173     }
0174     else if (PointerEventHandlersType == PointerEventHandlers.Highlight)
0175     {
0176         if (e.InitialPressMouseButton == MouseButton.Left)
0177         {
0178             IsMouseDown = false;
0179             if (eGetPosition(this).Equals(MouseDownPoint))
0180             {
0181                 this.Selection = null;
0182             }
0183         }
0184     }
0185     else if (PointerEventHandlersType == PointerEventHandlers.PanHighlight)
0186     {
0187         if (e.InitialPressMouseButton == MouseButton.Left)
0188         {
0189             IsMouseDown = false;
0190             if (CurrentMouseOperation == CurrentMouseOperations.Pan)
0191             {

```

```

01192             this.Cursor = new Cursor(StandardCursorType.Arrow);
01193         }
01194         if (e.GetPosition(this).Equals(MouseDownPoint))
01195         {
01196             this.Selection = null;
01197         }
01198     }
01199 }
01200 }
01201
01202 /// <summary>
01203 /// Default handler for the PointerMoved event (pan).
01204 /// </summary>
01205 /// <param name="sender"></param>
01206 /// <param name="e"></param>
01207 private void ControlPointerMoved(object sender, PointerEventArgs e)
01208 {
01209     if (IsMouseDown)
01210     {
01211         if (PointerEventHandlersType == PointerEventHandlers.Pan || (PointerEventHandlersType
01212 == PointerEventHandlers.PanHighlight && CurrentMouseOperation == CurrentMouseOperations.Pan))
01213         {
01214             Point point = e.GetPosition(this);
01215
01216             double deltaX = (-point.X + MouseDownPoint.X) / this.Bounds.Width *
01217 DisplayArea.Width;
01218             double deltaY = (-point.Y + MouseDownPoint.Y) / this.Bounds.Height *
01219 DisplayArea.Height;
01220
01221             Rect target = new Rect(new Point(this.MouseDownDisplayArea.X + deltaX,
01222 this.MouseDownDisplayArea.Y + deltaY), new Point(this.MouseDownDisplayArea.Right + deltaX,
01223 this.MouseDownDisplayArea.Bottom + deltaY));
01224
01225             SetDisplayAreaNowInternal(target);
01226             this.Cursor = new Cursor(StandardCursorType.SizeAll);
01227
01228         }
01229         else if (PointerEventHandlersType == PointerEventHandlers.Highlight ||
01230 (PointerEventHandlersType == PointerEventHandlers.PanHighlight && CurrentMouseOperation ==
01231 CurrentMouseOperations.Highlight))
01232         {
01233             Point point = e.GetPosition(this);
01234
01235             PointF pagePoint = new PointF((float)(point.X / this.Bounds.Width *
01236 DisplayArea.Width + DisplayArea.Left), (float)(point.Y / this.Bounds.Height * DisplayArea.Height +
01237 DisplayArea.Top));
01238
01239             MuPDFStructuredTextAddress? address =
01240 StructuredTextPage?.GetClosestHitAddress(pagePoint, false);
01241
01242             this.Selection = new MuPDFStructuredTextAddressSpan(this.Selection.Start,
01243 address);
01244
01245             if (address != null)
01246             {
01247                 this.Cursor = new Cursor(StandardCursorType.Ibeam);
01248             }
01249             else
01250             {
01251                 this.Cursor = new Cursor(StandardCursorType.Arrow);
01252             }
01253
01254         }
01255         else if (PointerEventHandlersType == PointerEventHandlers.Highlight ||
01256 PointerEventHandlersType == PointerEventHandlers.PanHighlight)
01257         {
01258             Point point = e.GetPosition(this);
01259
01260             PointF pagePoint = new PointF((float)(point.X / this.Bounds.Width *
01261 DisplayArea.Width + DisplayArea.Left), (float)(point.Y / this.Bounds.Height * DisplayArea.Height +
01262 DisplayArea.Top));
01263
01264             MuPDFStructuredTextAddress? address = StructuredTextPage?.GetHitAddress(pagePoint,
01265 false);
01266
01267             if (address != null)
01268             {
01269                 this.Cursor = new Cursor(StandardCursorType.Ibeam);
01270             }
01271             else
01272             {
01273                 this.Cursor = new Cursor(StandardCursorType.Arrow);
01274             }
01275
01276     }
01277 }
01278 }
```

```

01264         }
01265         else
01266     {
01267         this.Cursor = new Cursor(StandardCursorType.Arrow);
01268     }
01269 }
01270 }
01271
01272 /// <summary>
01273 /// Default handler for the PointerWheelChanged event (zoom in/out).
01274 /// </summary>
01275 /// <param name="sender"></param>
01276 /// <param name="e"></param>
01277 private void ControlPointerWheelChanged(object sender, PointerWheelEventArgs e)
01278 {
01279     if (ZoomEnabled)
01280     {
01281         ZoomStep(e.Delta.Y, e.GetPosition(this));
01282     }
01283 }
01284
01285 /// <summary>
01286 /// Compute the current value of the <see cref="Zoom"/> property.
01287 /// </summary>
01288 private void ComputeZoom()
01289 {
01290     //Take into account DPI scaling.
01291     double scale = (VisualRoot as ILayoutRoot).LayoutScaling;
01292     SetAndRaise(ZoomProperty, ref _Zoom, this.Bounds.Width / DisplayArea.Width * 72 / 96 *
scale);
01293 }
01294
01295 /// <summary>
01296 /// Draw the rendered document.
01297 /// </summary>
01298 /// <param name="context">The drawing context on which to draw.</param>
01299 public override void Render(DrawingContext context)
01300 {
01301     //Take into account DPI scaling.
01302     double scale = (VisualRoot as ILayoutRoot)?.LayoutScaling ?? 1;
01303
01304     context.FillRectangle(Background, this.Bounds);
01305
01306     //Page boundaries (used to draw the page background).
01307     double minX = Math.Max(PageSize.Left, DisplayArea.Left);
01308     double maxX = Math.Min(PageSize.Right, DisplayArea.Right);
01309     double minY = Math.Max(PageSize.Top, DisplayArea.Top);
01310     double maxY = Math.Min(PageSize.Bottom, DisplayArea.Bottom);
01311
01312
01313     if (IsViewerInitialized)
01314     {
01315         bool renderedDynamic = false;
01316
01317         //Check if someone is holding the mutex without blocking.
01318         if (RenderMutex.WaitOne(0))
01319         {
01320             //Check if the DynamicBitmaps are ready
01321             if (AreDynamicBitmapsReady)
01322             {
01323                 //Page background
01324                 context.FillRectangle(PageBackground, new Rect(new Point((minX -
DisplayArea.Left) / DisplayArea.Width * this.Bounds.Width, (minY - DisplayArea.Top) /
DisplayArea.Height * this.Bounds.Height), new Point((maxX - DisplayArea.Left) / DisplayArea.Width *
this.Bounds.Width, (maxY - DisplayArea.Top) / DisplayArea.Height * this.Bounds.Height)));
01325
01326                 //Draw the DynamicBitmaps.
01327                 for (int i = 0; i < DynamicImagesBounds.Length; i++)
01328                 {
01329                     context.DrawImage(DynamicBitmaps[i], new Rect(new Point(0, 0),
DynamicBitmaps[i].PixelSize.ToSize(1)), new Rect(DynamicImagesBounds[i].X0 / scale,
DynamicImagesBounds[i].Y0 / scale, DynamicImagesBounds[i].Width / scale,
DynamicImagesBounds[i].Height / scale));
01330                 }
01331
01332                 //Signal that we don't need to draw the static image.
01333                 renderedDynamic = true;
01334             }
01335
01336             //Release the mutex.
01337             RenderMutex.ReleaseMutex();
01338         }
01339
01340         //If the DynamicBitmaps have not been drawn, we fall back to drawing the static image
(which will probably be ugly and pixelated, but better than nothing).
01341         if (!renderedDynamic)
01342         {

```



```

01394             PathFigure quad = new PathFigure() { StartPoint = ll, IsClosed = true,
01395             IsFilled = true };
01396             quad.Segments.Add(new LineSegment() { Point = ul });
01397             quad.Segments.Add(new LineSegment() { Point = ur });
01398             quad.Segments.Add(new LineSegment() { Point = lr });
01399             selectionGeometry.Figures.Add(quad);
01400         }
01401     }
01402     context.DrawGeometry(this.SelectionBrush, null, selectionGeometry);
01403   }
01404 }
01405 }
01406 }
01407 }
01408 }
```

8.2 PDFRenderer.Properties.cs

```

00001 /*
00002  * MuPDFCore.MuPDFRenderer - A control to display documents in Avalonia using MuPDFCore.
00003  * Copyright (C) 2020 Giorgio Bianchini
00004
00005  This program is free software: you can redistribute it and/or modify
00006  it under the terms of the GNU Affero General Public License as
00007  published by the Free Software Foundation, version 3.
00008
00009  This program is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU Affero General Public License for more details.
00013
00014  You should have received a copy of the GNU Affero General Public License
00015  along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using Avalonia;
00019 using Avalonia.Controls;
00020 using Avalonia.Layout;
00021 using Avalonia.Media;
00022 using System;
00023 using System.Collections.Generic;
00024
00025 namespace MuPDFCore.MuPDFRenderer
00026 {
00027     public partial class PDFRenderer : Control
00028     {
00029         /// <summary>
00030         /// Defines the <see cref="RenderThreadCount"/> property.
00031         /// </summary>
00032         public static readonly DirectProperty<PDFRenderer, int> RenderThreadCountProperty =
00033             AvaloniaProperty.RegisterDirect<PDFRenderer, int>(nameof(RenderThreadCount), o =>
00034                 o.RenderThreadCount);
00035         /// <summary>
00036         /// Backing field for the <see cref="RenderThreadCount"/> property.
00037         /// </summary>
00038         /// Exposes the number of threads that the current instance is using to render the document.
00039         /// </summary>
00040         public int RenderThreadCount
00041         {
00042             get
00043             {
00044                 return _RenderThreadCount;
00045             }
00046
00047             private set
00048             {
00049                 SetAndRaise(RenderThreadCountProperty, ref _RenderThreadCount, value);
00050             }
00051         }
00052
00053         /// <summary>
00054         /// Defines the <see cref="PageNumber"/> property.
00055         /// </summary>
00056         public static readonly DirectProperty<PDFRenderer, int> PageNumberProperty =
00057             AvaloniaProperty.RegisterDirect<PDFRenderer, int>(nameof(PageNumber), o => o.PageNumber);
00058         /// <summary>
00059         /// Backing field for the <see cref="PageNumber"/> property.
00060         /// </summary>
00061         private int _PageNumber;
```

```

00061      /// <summary>
00062      /// Exposes the number of the page that the current instance is rendering. Read-only.
00063      /// </summary>
00064      public int PageNumber
00065      {
00066          get
00067          {
00068              return _PageNumber;
00069          }
00070
00071          private set
00072          {
00073              SetAndRaise(PageNumberProperty, ref _PageNumber, value);
00074          }
00075      }
00076
00077      /// <summary>
00078      /// Defines the <see cref="IsViewerInitialized"/> property.
00079      /// </summary>
00080      public static readonly DirectProperty<PDFRenderer, bool> IsViewerInitializedProperty =
AvaloniaProperty.RegisterDirect<PDFRenderer, bool>(nameof(IsViewerInitialized), o =>
o.IsViewerInitialized);
00081      /// <summary>
00082      /// Backing field for the <see cref="IsViewerInitialized"/> property.
00083      /// </summary>
00084      private bool _IsViewerInitialized = false;
00085      /// <summary>
00086      /// Whether the current instance has been initialised with a document to render or not.
00087      /// Read-only.
00088      /// </summary>
00089      public bool IsViewerInitialized
00090      {
00091          get
00092          {
00093              return _IsViewerInitialized;
00094          }
00095
00096          private set
00097          {
00098              SetAndRaise(IsViewerInitializedProperty, ref _IsViewerInitialized, value);
00099          }
00100
00101      /// <summary>
00102      /// Defines the <see cref="PageSize"/> property.
00103      /// </summary>
00104      public static readonly DirectProperty<PDFRenderer, Rect> PageSizeProperty =
AvaloniaProperty.RegisterDirect<PDFRenderer, Rect>(nameof(PageSize), o => o.PageSize);
00105      /// <summary>
00106      /// Backing field for the <see cref="PageSize"/> property.
00107      /// </summary>
00108      private Rect _PageSize;
00109      /// <summary>
00110      /// Exposes the size of the page that is drawn by the current instance (in page units).
00111      /// </summary>
00112      public Rect PageSize
00113      {
00114          get
00115          {
00116              return _PageSize;
00117          }
00118
00119          private set
00120          {
00121              SetAndRaise(PageSizeProperty, ref _PageSize, value);
00122          }
00123      }
00124
00125      /// <summary>
00126      /// Defines the <see cref="DisplayArea"/> property.
00127      /// </summary>
00128      public static readonly StyledProperty<Rect> DisplayAreaProperty =
AvaloniaProperty.Register<PDFRenderer, Rect>(nameof(DisplayArea));
00129      /// <summary>
00130      /// The region of the page (in page units) that is currently displayed by the current
00131      /// instance. This always has the same aspect ratio of the bounds of this control.
00132      /// When this is set, the value is sanitised so that the smallest rectangle with the correct
00133      /// aspect ratio containing the requested value is chosen.
00134      /// </summary>
00135      public Rect DisplayArea
00136      {
00137          get
00138          {
00139              return GetValue(DisplayAreaProperty);
00140          }
00141
00142          set
00143      }

```

```

00141         {
00142             double widthRatio = value.Width / (this.Bounds.Width);
00143             double heightRatio = value.Height / (this.Bounds.Height);
00144
00145             double containingWidth = Math.Max(widthRatio, heightRatio) * this.Bounds.Width;
00146             double containingHeight = Math.Max(widthRatio, heightRatio) * this.Bounds.Height;
00147
00148             double deltaW = (containingWidth - value.Width) * 0.5;
00149             double deltaH = (containingHeight - value.Height) * 0.5;
00150
00151             Rect newDispArea = new Rect(new Point(value.X - deltaW, value.Y - deltaH), new
00152             Point(value.Right + deltaW, value.Bottom + deltaH));
00153             SetValue(DisplayAreaProperty, newDispArea);
00154         }
00155     }
00156
00157     /// <summary>
00158     /// Defines the <see cref="ZoomIncrement"/> property.
00159     /// </summary>
00160     public static readonly StyledProperty<double> ZoomIncrementProperty =
00161         AvaloniaProperty.Register<PDFRenderer, double>(nameof(ZoomIncrement), Math.Pow(2, 1.0 / 3.0),
00162         defaultBindingMode: Avalonia.Data.BindingMode.TwoWay);
00163     /// <summary>
00164     /// Determines by how much the scale will be increased/decreased by the <see
00165     /// cref="ZoomStep(double, Point?)"/> method. Set this to a value smaller than 1 to invert the zoom
00166     /// in/out direction.
00167     /// </summary>
00168     public double ZoomIncrement
00169     {
00170         get { return GetValue(ZoomIncrementProperty); }
00171
00172         set
00173         {
00174             if (value <= 0)
00175             {
00176                 throw new ArgumentOutOfRangeException(nameof(ZoomIncrement), value, "The
00177                 ZoomIncrement must be greater than 0!");
00178             }
00179
00180             SetValue(ZoomIncrementProperty, value);
00181         }
00182
00183     /// <summary>
00184     /// Defines the <see cref="Background"/> property.
00185     /// </summary>
00186     public IBrush Background
00187     {
00188         get { return GetValue(BackgroundProperty); }
00189         set { SetValue(BackgroundProperty, value); }
00190     }
00191
00192     /// <summary>
00193     /// Defines the <see cref="PageBackground"/> property.
00194     /// </summary>
00195     public static readonly StyledProperty<IBrush> PageBackgroundProperty =
00196         AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(PageBackground));
00197     /// <summary>
00198     /// The background colour to use for the page drawn by the control.
00199     /// </summary>
00200     public IBrush PageBackground
00201     {
00202         get { return GetValue(PageBackgroundProperty); }
00203         set { SetValue(PageBackgroundProperty, value); }
00204     }
00205
00206     /// <summary>
00207     /// Defines the <see cref="Zoom"/> property.
00208     /// </summary>
00209     public static readonly DirectProperty<PDFRenderer, double> ZoomProperty =
00210         AvaloniaProperty.RegisterDirect<PDFRenderer, double>(nameof(Zoom), o => o.Zoom, (o, v) => o.Zoom = v,
00211         defaultBindingMode: Avalonia.Data.BindingMode.TwoWay);
00212     /// <summary>
00213     /// Backing field for the <see cref="Zoom"/> property.
00214     /// </summary>
00215     private double _Zoom;
00216     /// <summary>
00217     /// The current zoom level. Setting this will change the <see cref="DisplayArea"/>
00218     /// appropriately, zooming around the center of the <see cref="DisplayArea"/>.
00219     /// </summary>
00220     public double Zoom

```

```

00217      {
00218          get
00219          {
00220              return _Zoom;
00221          }
00222
00223          set
00224          {
00225              double actualZoom = value / _Zoom;
00226
00227              double currZoomX = FixedArea.Width / DisplayArea.Width * actualZoom;
00228              double currZoomY = FixedArea.Height / DisplayArea.Height * actualZoom;
00229
00230              double currWidth = FixedArea.Width / currZoomX;
00231              double currHeight = FixedArea.Height / currZoomY;
00232
00233
00234              Point pos = new Point(this.Bounds.Width * 0.5, this.Bounds.Height * 0.5);
00235
00236              double deltaW = currWidth - DisplayArea.Width;
00237              double deltaH = currHeight - DisplayArea.Height;
00238
00239              SetValue(DisplayAreaProperty, new Rect(new Point(DisplayArea.X - deltaW * pos.X /
00240 this.Bounds.Width, DisplayArea.Y - deltaH * pos.Y / this.Bounds.Height), new Point(DisplayArea.Right
+ deltaW * (1 - pos.X / this.Bounds.Width), DisplayArea.Bottom + deltaH * (1 - pos.Y /
00241 this.Bounds.Height))));
00242          }
00243
00244      /// <summary>
00245      /// Identifies the action to perform on pointer events.
00246      /// </summary>
00247      public enum PointerEventHandlers
00248      {
00249          /// <summary>
00250          /// Pointer events will be used to pan around the page.
00251          /// </summary>
00252          Pan,
00253
00254          /// <summary>
00255          /// Pointer events will be used to highlight text.
00256          /// </summary>
00257          Highlight,
00258
00259          /// <summary>
00260          /// Pointer events will be used to pan around the page or to highlight text, depending on
00261          /// where they start.
00262          /// </summary>
00263          PanHighlight,
00264
00265          /// <summary>
00266          /// Pointer events will be ignored. If you use this value, you will have to implement your
00267          /// own way to pan around the document by changing the <see cref="DisplayArea"/> or to select text.
00268          /// </summary>
00269          Custom
00270
00271      /// <summary>
00272      /// Defines the <see cref="PointerEventHandlersType"/> property.
00273      /// </summary>
00274      public static readonly StyledProperty<PointerEventHandlers> PointerEventHandlerTypeProperty =
00275      AvaloniaProperty.Register<PDFRenderer, PointerEventHandlers>(nameof(PointerEventHandlersType),
00276      PointerEventHandlers.PanHighlight);
00277
00278      /// <summary>
00279      /// Whether the default handlers for pointer events (which are used for panning around the
00280      /// page) should be enabled. If this is false, you will have to implement your own way to pan around the
00281      /// document by changing the <see cref="DisplayArea"/>.
00282      /// </summary>
00283      public PointerEventHandlers PointerEventHandlersType
00284      {
00285          get { return GetValue(PointerEventHandlerTypeProperty); }
00286          set { SetValue(PointerEventHandlerTypeProperty, value); }
00287      }
00288
00289      /// <summary>
00290      /// Defines the <see cref="ZoomEnabled"/> property.
00291      /// </summary>
00292      public static readonly StyledProperty<bool> ZoomEnabledProperty =
00293      AvaloniaProperty.Register<PDFRenderer, bool>(nameof(ZoomEnabled), true);
00294
00295      /// <summary>
00296      /// Whether the default handlers for pointer wheel events (which are used for zooming in/out)
00297      /// should be enabled. If this is false, you will have to implement your own way to zoom by changing the
00298      /// <see cref="DisplayArea"/>.
00299      /// </summary>
00300      public bool ZoomEnabled
00301      {
00302          get { return GetValue(ZoomEnabledProperty); }

```

```

00292         set { SetValue(ZoomEnabledProperty, value); }
00293     }
00294
00295     /// <summary>
00296     /// Defines the <see cref="Selection"/> property.
00297     /// </summary>
00298     public static readonly StyledProperty<MuPDFStructuredTextAddressSpan> SelectionProperty =
AvaloniaProperty.Register<PDFRenderer, MuPDFStructuredTextAddressSpan>(nameof(Selection), null);
00299     /// <summary>
00300     /// The start and end of the currently selected text.
00301     /// </summary>
00302     public MuPDFStructuredTextAddressSpan Selection
00303     {
00304         get { return GetValue(SelectionProperty); }
00305         set { SetValue(SelectionProperty, value); }
00306     }
00307
00308     /// <summary>
00309     /// Defines the <see cref="SelectionBrush"/> property.
00310     /// </summary>
00311     public static readonly StyledProperty<IBrush> SelectionBrushProperty =
AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(SelectionBrush), new
SolidColorBrush(Color.FromArgb(96, 86, 180, 233)));
00312     /// <summary>
00313     /// The colour used to highlight the <see cref="Selection"/>.
00314     /// </summary>
00315     public IBrush SelectionBrush
00316     {
00317         get { return GetValue(SelectionBrushProperty); }
00318         set { SetValue(SelectionBrushProperty, value); }
00319     }
00320
00321     /// <summary>
00322     /// Defines the <see cref="HighlightedRegions"/> property.
00323     /// </summary>
00324     public static readonly StyledProperty<IEnumerable<MuPDFStructuredTextAddressSpan>>
HighlightedRegionsProperty = AvaloniaProperty.Register<PDFRenderer,
IEnumerable<MuPDFStructuredTextAddressSpan>>(nameof(HighlightedRegions), null);
00325     /// <summary>
00326     /// A collection of highlighted regions, e.g. as a result of a text search.
00327     /// </summary>
00328     public IEnumerable<MuPDFStructuredTextAddressSpan> HighlightedRegions
00329     {
00330         get { return GetValue(HighlightedRegionsProperty); }
00331         set { SetValue(HighlightedRegionsProperty, value); }
00332     }
00333
00334     /// <summary>
00335     /// Defines the <see cref="HighlightBrush"/> property.
00336     /// </summary>
00337     public static readonly StyledProperty<IBrush> HighlightBrushProperty =
AvaloniaProperty.Register<PDFRenderer, IBrush>(nameof(HighlightBrush), new
SolidColorBrush(Color.FromArgb(96, 230, 159, 0)));
00338     /// <summary>
00339     /// The colour used to highlight the <see cref="HighlightedRegions"/>.
00340     /// </summary>
00341     public IBrush HighlightBrush
00342     {
00343         get { return GetValue(HighlightBrushProperty); }
00344         set { SetValue(HighlightBrushProperty, value); }
00345     }
00346 }
00347 }
```

8.3 RectTransition.cs

```

00001 /*
00002  * MuPDFCore.MuPDFRenderer - A control to display documents in Avalonia using MuPDFCore.
00003  * Copyright (C) 2020 Giorgio Bianchini
00004
00005  * This program is free software: you can redistribute it and/or modify
00006  * it under the terms of the GNU Affero General Public License as
00007  * published by the Free Software Foundation, version 3.
00008
00009  * This program is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU Affero General Public License for more details.
00013
00014  * You should have received a copy of the GNU Affero General Public License
00015  * along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
```

```

00018 using System;
00019 using System.Reactive.Linq;
00020
00021 namespace Avalonia.Animation
00022 {
00023     /// <summary>
00024     /// Transition class that handles <see cref="AvaloniaProperty"/> with <see cref="Rect"/> types.
00025     /// </summary>
00026     public class RectTransition : Transition<Rect>
00027     {
00028         /// <inheritdoc/>
00029         public override IObservable<Rect> DoTransition(IObservable<double> progress, Rect oldValue,
00030             Rect newValue)
00031         {
00032             return progress
00033                 .Select(p =>
00034                 {
00035                     var f = Easing.Ease(p);
00036
00037                     return new Rect((newValue.X - oldValue.X) * f + oldValue.X,
00038                         (newValue.Y - oldValue.Y) * f + oldValue.Y,
00039                         (newValue.Width - oldValue.Width) * f + oldValue.Width,
00040                         (newValue.Height - oldValue.Height) * f + oldValue.Height);
00041                 });
00042         }
00043     }

```

8.4 MuPDF.cs

```

00001 /*
00002     MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
00003     Copyright (C) 2020 Giorgio Bianchini
00004
00005     This program is free software: you can redistribute it and/or modify
00006     it under the terms of the GNU Affero General Public License as
00007     published by the Free Software Foundation, version 3.
00008
00009     This program is distributed in the hope that it will be useful,
00010     but WITHOUT ANY WARRANTY; without even the implied warranty of
00011     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012     GNU Affero General Public License for more details.
00013
00014     You should have received a copy of the GNU Affero General Public License
00015     along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using System;
00019 using System.IO;
00020 using System.IO.Pipes;
00021 using System.Runtime.InteropServices;
00022 using System.Text;
00023 using System.Threading.Tasks;
00024
00025 [assembly: System.Runtime.CompilerServices.InternalsVisibleTo("Tests")]
00026 namespace MuPDFCore
00027 {
00028     /// <summary>
00029     /// Exit codes returned by native methods describing various errors that can occur.
00030     /// </summary>
00031     public enum ExitCodes
00032     {
00033         /// <summary>
00034         /// An error occurred while creating the context object.
00035         /// </summary>
00036         ERR_CANNOT_CREATE_CONTEXT = 129,
00037
00038         /// <summary>
00039         /// An error occurred while registering the default document handlers with the context.
00040         /// </summary>
00041         ERR_CANNOT_REGISTER_HANDLERS = 130,
00042
00043         /// <summary>
00044         /// An error occurred while opening a file.
00045         /// </summary>
00046         ERR_CANNOT_OPEN_FILE = 131,
00047
00048         /// <summary>
00049         /// An error occurred while determining the total number of pages in the document.
00050         /// </summary>
00051         ERR_CANNOT_COUNT_PAGES = 132,
00052
00053         /// <summary>

```

```
00054     /// An error occurred while rendering the page.  
00055     /// </summary>  
00056     ERR_CANNOT_RENDER = 134,  
00057  
00058     /// <summary>  
00059     /// An error occurred while opening the stream.  
00060     /// </summary>  
00061     ERR_CANNOT_OPEN_STREAM = 135,  
00062  
00063     /// <summary>  
00064     /// An error occurred while loading the page.  
00065     /// </summary>  
00066     ERR_CANNOT_LOAD_PAGE = 136,  
00067  
00068     /// <summary>  
00069     /// An error occurred while computing the page bounds.  
00070     /// </summary>  
00071     ERR_CANNOT_COMPUTE_BOUNDS = 137,  
00072  
00073     /// <summary>  
00074     /// An error occurred while initialising the mutexes for the lock mechanism.  
00075     /// </summary>  
00076     ERR_CANNOT_INIT_MUTEX = 138,  
00077  
00078     /// <summary>  
00079     /// An error occurred while cloning the context.  
00080     /// </summary>  
00081     ERR_CANNOT_CLONE_CONTEXT = 139,  
00082  
00083     /// <summary>  
00084     /// An error occurred while saving the page to a raster image file.  
00085     /// </summary>  
00086     ERR_CANNOT_SAVE = 140,  
00087  
00088     /// <summary>  
00089     /// An error occurred while creating the output buffer.  
00090     /// </summary>  
00091     ERR_CANNOT_CREATE_BUFFER = 141,  
00092  
00093     /// <summary>  
00094     /// An error occurred while creating the document writer.  
00095     /// </summary>  
00096     ERR_CANNOT_CREATE_WRITER = 142,  
00097  
00098     /// <summary>  
00099     /// An error occurred while finalising the document file.  
00100     /// </summary>  
00101     ERR_CANNOT_CLOSE_DOCUMENT = 143,  
00102  
00103     /// <summary>  
00104     /// An error occurred while creating an empty structured text page.  
00105     /// </summary>  
00106     ERR_CANNOT_CREATE_PAGE = 144,  
00107  
00108     /// <summary>  
00109     /// An error occurred while populating the structured text page  
00110     /// </summary>  
00111     ERR_CANNOT_POPULATE_PAGE = 145,  
00112  
00113     /// <summary>  
00114     /// No error occurred. All is well.  
00115     /// </summary>  
00116     EXIT_SUCCESS = 0  
00117 }  
00118  
00119     /// <summary>  
00120     /// File types supported in input by the library.  
00121     /// </summary>  
00122     public enum InputFileTypes  
00123 {  
00124     /// <summary>  
00125     /// Portable Document Format.  
00126     /// </summary>  
00127     PDF = 0,  
00128  
00129     /// <summary>  
00130     /// XML Paper Specification document.  
00131     /// </summary>  
00132     XPS = 1,  
00133  
00134     /// <summary>  
00135     /// Comic book archive file (ZIP archive containing page scans).  
00136     /// </summary>  
00137     CBZ = 2,  
00138  
00139     /// <summary>  
00140     /// Portable Network Graphics format.
```

```
00141     /// </summary>
00142     PNG = 3,
00143
00144     /// <summary>
00145     /// Joint Photographic Experts Group image.
00146     /// </summary>
00147     JPEG = 4,
00148
00149     /// <summary>
00150     /// Bitmap image.
00151     /// </summary>
00152     BMP = 5,
00153
00154     /// <summary>
00155     /// Graphics Interchange Format.
00156     /// </summary>
00157     GIF = 6,
00158
00159     /// <summary>
00160     /// Tagged Image File Format.
00161     /// </summary>
00162     TIFF = 7,
00163
00164     /// <summary>
00165     /// Portable aNyMap graphics format.
00166     /// </summary>
00167     PNM = 8,
00168
00169     /// <summary>
00170     /// Portable Arbitrary Map graphics format.
00171     /// </summary>
00172     PAM = 9,
00173
00174     /// <summary>
00175     /// Electronic PUBLication document.
00176     /// </summary>
00177     EPUB = 10,
00178
00179     /// <summary>
00180     /// FictionBook document.
00181     /// </summary>
00182     FB2 = 11
00183 }
00184
00185     /// <summary>
00186     /// Raster image file types supported in output by the library.
00187     /// </summary>
00188     public enum RasterOutputFileTypes
00189     {
00190         /// <summary>
00191         /// Portable aNyMap graphics format.
00192         /// </summary>
00193         PNM = 0,
00194
00195         /// <summary>
00196         /// Portable Arbitrary Map graphics format.
00197         /// </summary>
00198         PAM = 1,
00199
00200         /// <summary>
00201         /// Portable Network Graphics format.
00202         /// </summary>
00203         PNG = 2,
00204
00205         /// <summary>
00206         /// PhotoShop Document format.
00207         /// </summary>
00208         PSD = 3
00209     };
00210
00211     /// <summary>
00212     /// Document file types supported in output by the library.
00213     /// </summary>
00214     public enum DocumentOutputFileTypes
00215     {
00216         /// <summary>
00217         /// Portable Document Format.
00218         /// </summary>
00219         PDF = 0,
00220
00221         /// <summary>
00222         /// Scalable Vector Graphics.
00223         /// </summary>
00224         SVG = 1,
00225
00226         /// <summary>
00227         /// Comic book archive format.
```

```
00228      /// </summary>
00229      CBZ = 2
00230  };
00231
00232  /// <summary>
00233  /// Pixel formats supported by the library.
00234  /// </summary>
00235  public enum PixelFormats
00236  {
00237      /// <summary>
00238      /// 24bpp RGB format.
00239      /// </summary>
00240      RGB = 0,
00241
00242      /// <summary>
00243      /// 32bpp RGBA format.
00244      /// </summary>
00245      RGBA = 1,
00246
00247      /// <summary>
00248      /// 24bpp BGR format.
00249      /// </summary>
00250      BGR = 2,
00251
00252      /// <summary>
00253      /// 32bpp BGRA format.
00254      /// </summary>
00255      BGRA = 3
00256  }
00257
00258  /// <summary>
00259  /// Possible document encryption states.
00260  /// </summary>
00261  public enum EncryptionState
00262  {
00263      /// <summary>
00264      /// The document is not encrypted.
00265      /// </summary>
00266      Unencrypted = 0,
00267
00268      /// <summary>
00269      /// The document is encrypted and a user password is necessary to render it.
00270      /// </summary>
00271      Encrypted = 1,
00272
00273      /// <summary>
00274      /// The document is encrypted and the correct user password has been supplied.
00275      /// </summary>
00276      Unlocked = 2
00277  }
00278
00279  /// <summary>
00280  /// Possible document restriction states.
00281  /// </summary>
00282  public enum RestrictionState
00283  {
00284      /// <summary>
00285      /// The document does not have any restrictions associated to it.
00286      /// </summary>
00287      Unrestricted = 0,
00288
00289      /// <summary>
00290      /// Some restrictions apply to the document. An owner password is required to remove these
00291      /// restrictions.
00292      /// </summary>
00293      Restricted = 1,
00294
00295      /// <summary>
00296      /// The document had some restrictions and the correct owner password has been supplied.
00297      /// </summary>
00298      Unlocked = 2
00299  }
00300
00301  /// <summary>
00302  /// Document restrictions.
00303  /// </summary>
00304  public enum DocumentRestrictions
00305  {
00306      /// <summary>
00307      /// No operation is restricted.
00308      /// </summary>
00309      None = 0,
00310
00311      /// <summary>
00312      /// Printing the document is restricted.
00313      /// </summary>
00314      Print = 1,
```

```

00314     /// <summary>
00315     /// Copying the document is restricted.
00316     /// </summary>
00317     Copy = 2,
00318
00319     /// <summary>
00320     /// Editing the document is restricted.
00321     /// </summary>
00322     Edit = 4,
00323
00324     /// <summary>
00325     /// Annotating the document is restricted.
00326     /// </summary>
00327     Annotate = 8
00328 }
00329
00330     /// <summary>
00331     /// Password types.
00332     /// </summary>
00333     public enum PasswordTypes
00334     {
00335         /// <summary>
00336         /// No password.
00337         /// </summary>
00338         None = 0,
00339
00340         /// <summary>
00341         /// The password corresponds to the user password.
00342         /// </summary>
00343         User = 1,
00344
00345         /// <summary>
00346         /// The password corresponds to the owner password.
00347         /// </summary>
00348         Owner = 2
00349     }
00350
00351     /// <summary>
00352     /// A struct to hold information about the current rendering process and to abort rendering as
00353     /// needed.
00354     /// </summary>
00355     [StructLayout(LayoutKind.Sequential)]
00356     internal struct Cookie
00357     {
00358         public int abort;
00359         public int progress;
00360         public ulong progress_max;
00361         public int errors;
00362         public int incomplete;
00363     }
00364
00365     /// <summary>
00366     /// Holds a summary of the progress of the current rendering operation.
00367     /// </summary>
00368     public class RenderProgress
00369     {
00370         /// <summary>
00371         /// Holds the progress of a single thread.
00372         /// </summary>
00373         public struct ThreadRenderProgress
00374         {
00375             /// <summary>
00376             /// The current progress.
00377             /// </summary>
00378             public int Progress;
00379
00380             /// <summary>
00381             /// The maximum progress. If this is 0, this value could not be determined (yet).
00382             /// </summary>
00383             public long MaxProgress;
00384
00385             internal ThreadRenderProgress(int progress, ulong maxProgress)
00386             {
00387                 this.Progress = progress;
00388                 this.MaxProgress = (long)maxProgress;
00389             }
00390         }
00391
00392         /// <summary>
00393         /// Contains the progress of all the threads used in rendering the document.
00394         /// </summary>
00395         public ThreadRenderProgress[] ThreadRenderProgresses { get; private set; }
00396
00397         internal RenderProgress(ThreadRenderProgress[] threadRenderProgresses)
00398         {
00399             ThreadRenderProgresses = threadRenderProgresses;

```

```
00400         }
00401     }
00402
00403     /// <summary>
00404     /// An <see cref="IDisposable"/> wrapper around an <see cref="IntPtr"/> that frees the allocated
00405     /// memory when it is disposed.
00406     public class DisposableIntPtr : IDisposable
00407     {
00408         /// <summary>
00409         /// The pointer to the unmanaged memory.
00410         /// </summary>
00411         private readonly IntPtr InternalPointer;
00412
00413         /// <summary>
00414         /// The number of bytes that have been allocated, for adding memory pressure.
00415         /// </summary>
00416         private readonly long BytesAllocated = -1;
00417
00418         /// <summary>
00419         /// Create a new DisposableIntPtr.
00420         /// </summary>
00421         /// <param name="pointer">The pointer that should be freed upon disposing of this
00422         /// object.</param>
00423         public DisposableIntPtr(IntPtr pointer)
00424         {
00425             this.InternalPointer = pointer;
00426         }
00427
00428         /// <summary>
00429         /// Create a new DisposableIntPtr, adding memory pressure to the GC to account for the
00430         /// allocation of unmanaged memory.
00431         /// <param name="pointer">The pointer that should be freed upon disposing of this
00432         /// object.</param>
00433         /// <param name="bytesAllocated">The number of bytes that have been allocated, for adding
00434         /// memory pressure.</param>
00435         public DisposableIntPtr(IntPtr pointer, long bytesAllocated)
00436         {
00437             this.InternalPointer = pointer;
00438             this.BytesAllocated = bytesAllocated;
00439
00440             if (BytesAllocated > 0)
00441             {
00442                 GC.AddMemoryPressure(bytesAllocated);
00443             }
00444
00445             private bool disposedValue;
00446
00447             ///<inheritDoc/>
00448             protected virtual void Dispose(bool disposing)
00449             {
00450                 if (!disposedValue)
00451                 {
00452                     Marshal.FreeHGlobal(InternalPointer);
00453
00454                     if (BytesAllocated > 0)
00455                     {
00456                         GC.RemoveMemoryPressure(BytesAllocated);
00457
00458                     disposedValue = true;
00459                 }
00460             }
00461
00462             ///<inheritDoc/>
00463             ~DisposableIntPtr()
00464             {
00465                 Dispose(disposing: false);
00466             }
00467
00468             ///<inheritDoc/>
00469             public void Dispose()
00470             {
00471                 Dispose(disposing: true);
00472                 GC.SuppressFinalize(this);
00473             }
00474
00475             /// <summary>
00476             /// The exception that is thrown when a MuPDF operation fails.
00477             /// </summary>
00478             public class MuPDFException : Exception
00479             {
00480                 /// <summary>
00481                 /// The <see cref="ExitCodes"/> returned by the native function.
```

```
00482     /// </summary>
00483     public readonly ExitCodes ErrorCode;
00484
00485     internal MuPDFException(string message, ExitCodes errorCode) : base(message)
00486     {
00487         this.ErrorCode = errorCode;
00488     }
00489 }
00490
00491 /// <summary>
00492 /// The exception that is thrown when an attempt is made to render an encrypted document without
00493 supplying the required password.
00494 /// </summary>
00495 public class DocumentLockedException : Exception
00496 {
00497     internal DocumentLockedException(string message) : base(message) { }
00498 }
00499
00500 /// <summary>
00501 /// A class to simplify passing a string to the MuPDF C library with the correct encoding.
00502 /// </summary>
00503 internal class UTF8EncodedString : IDisposable
00504 {
00505     private bool disposedValue;
00506
00507     /// <summary>
00508     /// The address of the bytes encoding the string in unmanaged memory.
00509     /// </summary>
00510     public IntPtr Address { get; }
00511
00512     /// <summary>
00513     /// Create a null-terminated, UTF-8 encoded string in unmanaged memory.
00514     /// </summary>
00515     public UTF8EncodedString(string text)
00516     {
00517         byte[] data = System.Text.Encoding.UTF8.GetBytes(text);
00518
00519         IntPtr dataHolder = Marshal.AllocHGlobal(data.Length + 1);
00520         Marshal.Copy(data, 0, dataHolder, data.Length);
00521         Marshal.WriteByte(dataHolder, data.Length, 0);
00522
00523         this.Address = dataHolder;
00524     }
00525
00526     protected virtual void Dispose(bool disposing)
00527     {
00528         if (!disposedValue)
00529         {
00530             Marshal.FreeHGlobal(Address);
00531             disposedValue = true;
00532         }
00533     }
00534
00535     ~UTF8EncodedString()
00536     {
00537         Dispose(disposing: false);
00538     }
00539
00540     public void Dispose()
00541     {
00542         Dispose(disposing: true);
00543         GC.SuppressFinalize(this);
00544     }
00545 }
00546
00547 /// <summary>
00548 /// EventArgs for the <see cref="MuPDF.StandardOutputMessage"/> and <see
00549 cref="MuPDF.StandardErrorMessage"/> events.
00550 /// </summary>
00551 public class MessageEventArgs : EventArgs
00552 {
00553     /// <summary>
00554     /// The message that has been logged.
00555     /// </summary>
00556     public string Message { get; }
00557
00558     /// Create a new <see cref="MessageEventArgs"/> instance.
00559     /// </summary>
00560     /// <param name="message">The message that has been logged.</param>
00561     public MessageEventArgs(string message)
00562     {
00563         this.Message = message;
00564     }
00565 }
00566
```



```

00645             {
00646                 try
00647                 {
00648                     client.Connect(100);
00649                     break;
00650                 }
00651             catch { }
00652         }
00653
00654         using (StreamReader reader = new StreamReader(client))
00655         {
00656             while (true)
00657             {
00658                 string message = reader.ReadLine();
00659
00660                 if (!string.IsNullOrEmpty(message))
00661                 {
00662                     StandardOutputMessage?.Invoke(null, new
00663                     MessageEventArgs(message));
00664                 }
00665             }
00666         }
00667     });
00668
00669     // Start stderr pipe (this is actually a socket)
00670     _ = Task.Run(() =>
00671     {
00672         using (NamedPipeClientStream client = new NamedPipeClientStream(pipeName +
00673         "-err"))
00674         {
00675             while (true)
00676             {
00677                 try
00678                 {
00679                     client.Connect(100);
00680                     break;
00681                 }
00682             catch { }
00683         }
00684
00685         using (StreamReader reader = new StreamReader(client))
00686         {
00687             while (true)
00688             {
00689                 string message = reader.ReadLine();
00690
00691                 if (!string.IsNullOrEmpty(message))
00692                 {
00693                     StandardErrorMessage?.Invoke(null, new MessageEventArgs(message));
00694                 }
00695             }
00696         }
00697     });
00698
00699 });
00700
00701     await redirectOutputTask;
00702
00703     ConsoleOut = Console.Out;
00704     ConsoleErr = Console.Error;
00705
00706     ConsoleColor fg = Console.ForegroundColor;
00707     ConsoleColor bg = Console.BackgroundColor;
00708
00709     Console.ResetColor();
00710
00711     DefaultForeground = Console.ForegroundColor;
00712     DefaultBackground = Console.BackgroundColor;
00713
00714     Console.ForegroundColor = fg;
00715     Console.BackgroundColor = bg;
00716
00717     Console.SetOut(new FileDescriptorTextWriter(Console.Out.Encoding, StdOutFD));
00718     Console.SetError(new FileDescriptorTextWriter(Console.Error.Encoding, StdErrFD));
00719     }
00720   }
00721
00722     private static async Task RedirectOutputWindows()
00723     {
00724         if (StdOutFD < 0 && StdErrFD < 0)
00725         {
00726             string pipeName = "MuPDFCore-" + Guid.NewGuid().ToString();
00727
00728             Task redirectOutputTask = System.Threading.Tasks.Task.Run(() =>
00729             {

```

```
00730             NativeMethods.RedirectOutput(out StdOutFD, out StdErrFD, "\\\\.\\" + pipeName + "-out", "\\\\.\\" + pipeName + "-err");
00731         });
00732 
00733         // Start stdout pipe
00734         _ = Task.Run(() =>
00735     {
00736         using (NamedPipeClientStream client = new NamedPipeClientStream(pipeName +
00737             "-out"))
00738         {
00739             while (true)
00740             {
00741                 try
00742                 {
00743                     client.Connect(100);
00744                     break;
00745                 }
00746                 catch { }
00747             }
00748 
00749             using (StreamReader reader = new StreamReader(client))
00750             {
00751                 while (true)
00752                 {
00753                     string message = reader.ReadLine();
00754 
00755                     if (!string.IsNullOrEmpty(message))
00756                     {
00757                         StandardOutputMessage?.Invoke(null, new
00758                             MessageEventArgs(message));
00759                     }
00760                 }
00761             });
00762 
00763         // Start stderr pipe
00764         _ = Task.Run(() =>
00765     {
00766         using (NamedPipeClientStream client = new NamedPipeClientStream(pipeName +
00767             "-err"))
00768         {
00769             while (true)
00770             {
00771                 try
00772                 {
00773                     client.Connect(100);
00774                     break;
00775                 }
00776                 catch { }
00777             }
00778 
00779             using (StreamReader reader = new StreamReader(client))
00780             {
00781                 while (true)
00782                 {
00783                     string message = reader.ReadLine();
00784 
00785                     if (!string.IsNullOrEmpty(message))
00786                     {
00787                         StandardErrorMessage?.Invoke(null, new MessageEventArgs(message));
00788                     }
00789                 }
00790             }
00791         });
00792 
00793     });
00794 
00795     await redirectOutputTask;
00796 
00797     ConsoleOut = Console.Out;
00798     ConsoleErr = Console.Error;
00799 
00800     ConsoleColor fg = Console.ForegroundColor;
00801     ConsoleColor bg = Console.BackgroundColor;
00802 
00803     Console.ResetColor();
00804 
00805     DefaultForeground = Console.ForegroundColor;
00806     DefaultBackground = Console.BackgroundColor;
00807 
00808     Console.ForegroundColor = fg;
00809     Console.BackgroundColor = bg;
00810 
00811     Console.SetOut(new FileDescriptorTextWriter(Console.Out.Encoding, StdOutFD));
00812     Console.SetError(new FileDescriptorTextWriter(Console.Error.Encoding, StdErrFD));
```

```
00813         }
00814     }
00815
00816     /// <summary>
00817     /// Reset the default standard output and error streams for the native MuPDF library.
00818     /// </summary>
00819     public static void ResetOutput()
00820     {
00821         lock (CleanupLock)
00822         {
00823             if (StdOutFD >= 0 && StdErrFD >= 0)
00824             {
00825                 NativeMethods.ResetOutput(StdOutFD, StdErrFD);
00826
00827                 Console.SetOut(ConsoleOut);
00828                 Console.SetError(ConsoleErr);
00829
00830                 StdOutFD = -1;
00831                 StdErrFD = -1;
00832
00833             if
00834             (!System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
00835             {
00836                 File.Delete(PipeName + "-out");
00837                 File.Delete(PipeName + "-err");
00838             }
00839         }
00840     }
00841
00842     internal class FileDescriptorTextWriter : TextWriter
00843     {
00844         public override Encoding Encoding { get; }
00845         private int FileDescriptor { get; }
00846
00847         public FileDescriptorTextWriter(Encoding encoding, int fileDescriptor)
00848         {
00849             this.Encoding = encoding;
00850             this.FileDescriptor = fileDescriptor;
00851         }
00852
00853         public override void Write(string value)
00854         {
00855             StringBuilder sb = new StringBuilder();
00856
00857             if (Console.ForegroundColor != DefaultForeground || Console.BackgroundColor != DefaultBackground)
00858             {
00859                 sb.Append(" [");
00860             }
00861
00862             if (Console.ForegroundColor != DefaultForeground)
00863             {
00864                 switch (Console.ForegroundColor)
00865                 {
00866                     case ConsoleColor.Black:
00867                         sb.Append("30");
00868                         break;
00869                     case ConsoleColor.DarkRed:
00870                         sb.Append("31");
00871                         break;
00872                     case ConsoleColor.DarkGreen:
00873                         sb.Append("32");
00874                         break;
00875                     case ConsoleColor.DarkYellow:
00876                         sb.Append("33");
00877                         break;
00878                     case ConsoleColor.DarkBlue:
00879                         sb.Append("34");
00880                         break;
00881                     case ConsoleColor.DarkMagenta:
00882                         sb.Append("35");
00883                         break;
00884                     case ConsoleColor.DarkCyan:
00885                         sb.Append("36");
00886                         break;
00887                     case ConsoleColor.Gray:
00888                         sb.Append("37");
00889                         break;
00890                     case ConsoleColor.DarkGray:
00891                         sb.Append("90");
00892                         break;
00893                     case ConsoleColor.Red:
00894                         sb.Append("91");
00895                         break;
00896                     case ConsoleColor.Green:
00897                         sb.Append("92");
```

```
00898             break;
00899         case ConsoleColor.Yellow:
00900             sb.Append("93");
00901             break;
00902         case ConsoleColor.Blue:
00903             sb.Append("94");
00904             break;
00905         case ConsoleColor.Magenta:
00906             sb.Append("95");
00907             break;
00908         case ConsoleColor.Cyan:
00909             sb.Append("96");
00910             break;
00911         case ConsoleColor.White:
00912             sb.Append("97");
00913             break;
00914     }
00915 }
00916
00917 if (Console.ForegroundColor != DefaultForeground && Console.BackgroundColor !=
DefaultBackground)
00918 {
00919     sb.Append(";");
00920 }
00921
00922 if (Console.BackgroundColor != DefaultBackground)
00923 {
00924     switch (Console.BackgroundColor)
00925     {
00926         case ConsoleColor.Black:
00927             sb.Append("40");
00928             break;
00929         case ConsoleColor.DarkRed:
00930             sb.Append("41");
00931             break;
00932         case ConsoleColor.DarkGreen:
00933             sb.Append("42");
00934             break;
00935         case ConsoleColor.DarkYellow:
00936             sb.Append("43");
00937             break;
00938         case ConsoleColor.DarkBlue:
00939             sb.Append("44");
00940             break;
00941         case ConsoleColor.DarkMagenta:
00942             sb.Append("45");
00943             break;
00944         case ConsoleColor.DarkCyan:
00945             sb.Append("46");
00946             break;
00947         case ConsoleColor.Gray:
00948             sb.Append("47");
00949             break;
00950         case ConsoleColor.DarkGray:
00951             sb.Append("100");
00952             break;
00953         case ConsoleColor.Red:
00954             sb.Append("101");
00955             break;
00956         case ConsoleColor.Green:
00957             sb.Append("102");
00958             break;
00959         case ConsoleColor.Yellow:
00960             sb.Append("103");
00961             break;
00962         case ConsoleColor.Blue:
00963             sb.Append("104");
00964             break;
00965         case ConsoleColor.Magenta:
00966             sb.Append("105");
00967             break;
00968         case ConsoleColor.Cyan:
00969             sb.Append("106");
00970             break;
00971         case ConsoleColor.White:
00972             sb.Append("107");
00973             break;
00974     }
00975 }
00976
00977 if (Console.ForegroundColor != DefaultForeground || Console.BackgroundColor !=
DefaultBackground)
00978 {
00979     sb.Append("m");
00980 }
00981
00982 sb.Append(value);
```

```

00983
00984         if (Console.ForegroundColor != DefaultForeground || Console.BackgroundColor != DefaultBackground)
00985             {
00986                 sb.Append(" [ ");
00987             }
00988
00989         if (Console.ForegroundColor != DefaultForeground)
00990             {
00991                 sb.Append("39");
00992             }
00993
00994         if (Console.ForegroundColor != DefaultForeground && Console.BackgroundColor != DefaultBackground)
00995             {
00996                 sb.Append(";");
00997             }
00998
00999         if (Console.BackgroundColor != DefaultBackground)
01000             {
01001                 sb.Append("49");
01002             }
01003
01004         if (Console.ForegroundColor != DefaultForeground || Console.BackgroundColor != DefaultBackground)
01005             {
01006                 sb.Append("m");
01007             }
01008
01009         NativeMethods.WriteToFileDescriptor(FileDescriptor, sb.ToString(), sb.Length);
01010     }
01011
01012     public override void Write(char value)
01013     {
01014         Write(value.ToString());
01015     }
01016
01017     public override void Write(char[] buffer)
01018     {
01019         Write(new string(buffer));
01020     }
01021
01022     public override void Write(char[] buffer, int index, int count)
01023     {
01024         Write(new string(buffer, index, count));
01025     }
01026
01027     public override void WriteLine(string value)
01028     {
01029         Write(value);
01030         WriteLine();
01031     }
01032 }
01033 }
01034
01035 /// <summary>
01036 /// Native methods.
01037 /// </summary>
01038 internal static class NativeMethods
01039 {
01040     /// <summary>
01041     /// Create a MuPDF context object with the specified store size.
01042     /// </summary>
01043     /// <param name="store_size">Maximum size in bytes of the resource store.</param>
01044     /// <param name="out_ctx">A pointer to the native context object.</param>
01045     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01046     /// occurred.</returns>
01047     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01048     internal static extern int CreateContext(ulong store_size, ref IntPtr out_ctx);
01049
01050     /// <summary>
01051     /// Free a context and its global store.
01052     /// </summary>
01053     /// <param name="ctx">A pointer to the native context to free.</param>
01054     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01055     /// occurred.</returns>
01056     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01057     internal static extern int DisposeContext(IntPtr ctx);
01058
01059     /// <summary>
01060     /// Evict items from the store until the total size of the objects in the store is reduced to
01061     /// a given percentage of its current size.
01062     /// </summary>
01063     /// <param name="ctx">The context whose store should be shrunk.</param>
01064     /// <param name="perc">Fraction of current size to reduce the store to.</param>
01065     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01066     /// occurred.</returns>

```

```

01063     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01064     internal static extern int ShrinkStore(IntPtr ctx, uint perc);
01065
01066     /// <summary>
01067     /// Evict every item from the store.
01068     /// </summary>
01069     /// <param name="ctx">The context whose store should be emptied.</param>
01070     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01071     /// occurred.</returns>
01072     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01073     internal static extern void EmptyStore(IntPtr ctx);
01074
01075     /// <summary>
01076     /// Get the current size of the store.
01077     /// </summary>
01078     /// <param name="ctx">The context whose store's size should be determined.</param>
01079     /// <returns>The current size in bytes of the store.</returns>
01080     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01081     internal static extern ulong GetCurrentStoreSize(IntPtr ctx);
01082
01083     /// <summary>
01084     /// Get the maximum size of the store.
01085     /// </summary>
01086     /// <param name="ctx">The context whose store's maximum size should be determined.</param>
01087     /// <returns>The maximum size in bytes of the store.</returns>
01088     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01089     internal static extern ulong GetMaxStoreSize(IntPtr ctx);
01090
01091     /// <summary>
01092     /// Create a display list from a page.
01093     /// </summary>
01094     /// <param name="ctx">A pointer to the context used to create the document.</param>
01095     /// <param name="page">A pointer to the page that should be used to create the display
01096     /// list.</param>
01097     /// <param name="annotations">An integer indicating whether annotations should be included in
01098     /// the display list (1) or not (any other value).</param>
01099     /// <param name="out_display_list">A pointer to the newly-created display list.</param>
01100     /// <param name="out_x0">The left coordinate of the display list's bounds.</param>
01101     /// <param name="out_y0">The top coordinate of the display list's bounds.</param>
01102     /// <param name="out_x1">The right coordinate of the display list's bounds.</param>
01103     /// <param name="out_y1">The bottom coordinate of the display list's bounds.</param>
01104     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01105     /// occurred.</returns>
01106     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01107     internal static extern int GetDisplayList(IntPtr ctx, IntPtr page, int annotations, ref IntPtr
01108     out_display_list, ref float out_x0, ref float out_y0, ref float out_x1, ref float out_y1);
01109
01110     /// <summary>
01111     /// Free a display list.
01112     /// </summary>
01113     /// <param name="ctx">The context that was used to create the display list.</param>
01114     /// <param name="list">The display list to dispose.</param>
01115     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01116     /// occurred.</returns>
01117     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01118     internal static extern int DisposeDisplayList(IntPtr ctx, IntPtr list);
01119
01120     /// <summary>
01121     /// Create a new document from a stream.
01122     /// </summary>
01123     /// <param name="ctx">The context to which the document will belong.</param>
01124     /// <param name="data">A pointer to a byte array containing the data that makes up the
01125     /// document.</param>
01126     /// <param name="data_length">The length in bytes of the data that makes up the
01127     /// document.</param>
01128     /// <param name="file_type">The type (extension) of the document.</param>
01129     /// <param name="get_image_resolution">If this is not 0, try opening the stream as an image
01130     /// and return the actual resolution (in DPI) of the image. Otherwise (or if trying to open the stream as
01131     /// an image fails), the returned resolution will be -1.</param>
01132     /// <param name="out_doc">The newly created document.</param>
01133     /// <param name="out_str">The newly created stream (so that it can be disposed later).</param>
01134     /// <param name="out_page_count">The number of pages in the document.</param>
01135     /// <param name="out_image_xres">If the document is an image file, the horizontal resolution
01136     /// of the image.</param>
01137     /// <param name="out_image_yres">If the document is an image file, the vertical resolution of
01138     /// the image.</param>
01139     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01140     /// occurred.</returns>
01141     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01142     internal static extern int CreateDocumentFromStream(IntPtr ctx, IntPtr data, ulong
01143     data_length, string file_type, int get_image_resolution, ref IntPtr out_doc, ref IntPtr out_str, ref
01144     int out_page_count, ref float out_image_xres, ref float out_image_yres);
01145
01146     /// <summary>
01147     /// Create a new document from a file name.
01148     /// </summary>
01149     /// <param name="ctx">The context to which the document will belong.</param>

```

```

01135     /// <param name="file_name">The path of the file to open, UTF-8 encoded.</param>
01136     /// <param name="get_image_resolution">If this is not 0, try opening the file as an image and
01137     // return the actual resolution (in DPI) of the image. Otherwise (or if trying to open the file as an
01138     // image fails), the returned resolution will be -1.</param>
01139     /// <param name="out_doc">The newly created document.</param>
01140     /// <param name="out_page_count">The number of pages in the document.</param>
01141     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01142     // occurred.</returns>
01143     /// <param name="out_image_xres">If the document is an image file, the horizontal resolution
01144     // of the image.</param>
01145     /// <param name="out_image_yres">If the document is an image file, the vertical resolution of
01146     // the image.</param>
01147     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01148     internal static extern int CreateDocumentFromFile(IntPtr ctx, IntPtr file_name, int
01149     get_image_resolution, ref IntPtr out_doc, ref int out_page_count, ref float out_image_xres, ref float
01150     out_image_yres);

01151     /// <summary>
01152     // Free a stream and its associated resources.
01153     /// </summary>
01154     /// <param name="ctx">The context that was used while creating the stream.</param>
01155     /// <param name="str">The stream to free.</param>
01156     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01157     // occurred.</returns>
01158     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01159     internal static extern int DisposeStream(IntPtr ctx, IntPtr str);

01160     /// <summary>
01161     // Free a document and its associated resources.
01162     /// </summary>
01163     /// <param name="ctx">The context that was used in creating the document.</param>
01164     /// <param name="doc">The document to free.</param>
01165     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01166     // occurred.</returns>
01167     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01168     internal static extern int DisposeDocument(IntPtr ctx, IntPtr doc);

01169     /// <summary>
01170     // Render (part of) a display list to an array of bytes starting at the specified pointer.
01171     /// </summary>
01172     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01173     /// <param name="list">The display list to render.</param>
01174     /// <param name="x0">The left coordinate in page units of the region of the display list that
01175     // should be rendererd.</param>
01176     /// <param name="y0">The top coordinate in page units of the region of the display list that
01177     // should be rendererd.</param>
01178     /// <param name="x1">The right coordinate in page units of the region of the display list that
01179     // should be rendererd.</param>
01180     /// <param name="y1">The bottom coordinate in page units of the region of the display list
01181     // that should be rendererd.</param>
01182     /// <param name="zoom">How much the specified region should be scaled when rendering. This
01183     // determines the size in pixels of the rendered image.</param>
01184     /// <param name="colorFormat">The pixel data format.</param>
01185     /// <param name="pixel_storage">A pointer indicating where the pixel bytes will be written.
01186     // There must be enough space available!</param>
01187     /// <param name="cookie">A pointer to a cookie object that can be used to track progress
01188     // and/or abort rendering. Can be null.</param>
01189     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01190     // occurred.</returns>
01191     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01192     internal static extern int RenderSubDisplayList(IntPtr ctx, IntPtr list, float x0, float y0,
01193     float x1, float y1, float zoom, int colorFormat, IntPtr pixel_storage, IntPtr cookie);

01194     /// <summary>
01195     // Load a page from a document.
01196     /// </summary>
01197     /// <param name="ctx">The context to which the document belongs.</param>
01198     /// <param name="doc">The document from which the page should be extracted.</param>
01199     /// <param name="page_number">The page number.</param>
01200     /// <param name="out_page">The newly extracted page.</param>
01201     /// <param name="out_x">The left coordinate of the page's bounds.</param>
01202     /// <param name="out_y">The top coordinate of the page's bounds.</param>
01203     /// <param name="out_w">The width of the page.</param>
01204     /// <param name="out_h">The height of the page.</param>
01205     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01206     // occurred.</returns>
01207     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01208     internal static extern int LoadPage(IntPtr ctx, IntPtr doc, int page_number, ref IntPtr
01209     out_page, ref float out_x, ref float out_y, ref float out_w, ref float out_h);

01210     /// <summary>
01211     // Free a page and its associated resources.
01212     /// </summary>
01213     /// <param name="ctx">The context to which the document containing the page belongs.</param>
01214     /// <param name="page">The page to free.</param>
01215     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01216     // occurred.</returns>

```

```

01201     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01202     internal static extern int DisposePage(IntPtr ctx, IntPtr page);
01203
01204     /// <summary>
01205     /// Create cloned contexts that can be used in multithreaded rendering.
01206     /// </summary>
01207     /// <param name="ctx">The original context to clone.</param>
01208     /// <param name="count">The number of cloned contexts to create.</param>
01209     /// <param name="out_contexts">An array of pointers to the cloned contexts.</param>
01210     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01211     occurred.</returns>
01212     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01213     internal static extern int CloneContext(IntPtr ctx, int count, IntPtr out_contexts);
01214
01215     /// <summary>
01216     /// Save (part of) a display list to an image file in the specified format.
01217     /// </summary>
01218     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01219     /// <param name="list">The display list to render.</param>
01220     /// <param name="x0">The left coordinate in page units of the region of the display list that
01221     should be rendererd.</param>
01222     /// <param name="y0">The top coordinate in page units of the region of the display list that
01223     should be rendererd.</param>
01224     /// <param name="x1">The right coordinate in page units of the region of the display list that
01225     should be rendererd.</param>
01226     /// <param name="y1">The bottom coordinate in page units of the region of the display list
01227     that should be rendererd.</param>
01228     /// <param name="zoom">How much the specified region should be scaled when rendering. This
01229     determines the size in pixels of the rendered image.</param>
01230     /// <param name="colorFormat">The pixel data format.</param>
01231     /// <param name="file_name">The path to the output file, UTF-8 encoded.</param>
01232     /// <param name="output_format">An integer equivalent to <see cref="RasterOutputFileTypes"/>
01233     specifying the output format.</param>
01234     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01235     occurred.</returns>
01236     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01237     internal static extern int SaveImage(IntPtr ctx, IntPtr list, float x0, float y0, float x1,
01238     float y1, float zoom, int colorFormat, IntPtr file_name, int output_format);
01239
01240     /// <summary>
01241     /// Write (part of) a display list to an image buffer in the specified format.
01242     /// </summary>
01243     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01244     /// <param name="list">The display list to render.</param>
01245     /// <param name="x0">The left coordinate in page units of the region of the display list that
01246     should be rendererd.</param>
01247     /// <param name="y0">The top coordinate in page units of the region of the display list that
01248     should be rendererd.</param>
01249     /// <param name="x1">The right coordinate in page units of the region of the display list that
01250     should be rendererd.</param>
01251     /// <param name="y1">The bottom coordinate in page units of the region of the display list
01252     that should be rendererd.</param>
01253     /// <param name="zoom">How much the specified region should be scaled when rendering. This
01254     determines the size in pixels of the rendered image.</param>
01255     /// <param name="colorFormat">The pixel data format.</param>
01256     /// <param name="output_format">An integer equivalent to <see cref="RasterOutputFileTypes"/>
01257     specifying the output format.</param>
01258     /// <param name="out_buffer">The address of the buffer on which the data has been written
01259     (only useful for disposing the buffer later).</param>
01260     /// <param name="out_data">The address of the byte array where the data has been actually
01261     written.</param>
01262     /// <param name="out_length">The length in bytes of the image data.</param>
01263     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01264     occurred.</returns>
01265     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01266     internal static extern int WriteImage(IntPtr ctx, IntPtr list, float x0, float y0, float x1,
01267     float y1, float zoom, int colorFormat, int output_format, ref IntPtr out_buffer, ref IntPtr out_data,
01268     ref ulong out_length);
01269
01270     /// <summary>
01271     /// Free a native buffer and its associated resources.
01272     /// </summary>
01273     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01274     /// <param name="buf">The buffer to free.</param>
01275     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01276     occurred.</returns>
01277     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01278     internal static extern int DisposeBuffer(IntPtr ctx, IntPtr buf);
01279
01280     /// <summary>
01281     /// Create a new document writer object.
01282     /// </summary>
01283     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01284     /// <param name="file_name">The name of file that will hold the writer's output, UTF-8
01285     encoded.</param>
01286     /// <param name="format">An integer equivalent to <see cref="DocumentOutputFileTypes"/>
01287     specifying the output format.</param>

```

```

01265     /// <param name="out_document_writer">A pointer to the new document writer object.</param>
01266     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01267     occurred.</returns>
01268     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01269     internal static extern int CreateDocumentWriter(IntPtr ctx, IntPtr file_name, int format, ref
01270     IntPtr out_document_writer);
01271 
01272     /// <summary>
01273     /// Render (part of) a display list as a page in the specified document writer.
01274     /// </summary>
01275     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01276     /// <param name="list">The display list to render.</param>
01277     /// <param name="x0">The left coordinate in page units of the region of the display list that
01278     should be rendererd.</param>
01279     /// <param name="y0">The top coordinate in page units of the region of the display list that
01280     should be rendererd.</param>
01281     /// <param name="x1">The right coordinate in page units of the region of the display list that
01282     should be rendererd.</param>
01283     /// <param name="y1">The bottom coordinate in page units of the region of the display list
01284     that should be rendererd.</param>
01285     /// <param name="zoom">How much the specified region should be scaled when rendering. This
01286     will determine the final size of the page.</param>
01287     /// <param name="writ">The document writer on which the page should be written.</param>
01288     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01289     occurred.</returns>
01290     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01291     internal static extern int WriteSubDisplayListAsPage(IntPtr ctx, IntPtr list, float x0, float
01292     y0, float x1, float y1, float zoom, IntPtr writ);
01293 
01294     /// <summary>
01295     /// Finalise a document writer, closing the file and freeing all resources.
01296     /// </summary>
01297     /// <param name="ctx">The context that was used to create the document writer.</param>
01298     /// <param name="writ">The document writer to finalise.</param>
01299     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01300     occurred.</returns>
01301     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01302     internal static extern int FinalizeDocumentWriter(IntPtr ctx, IntPtr writ);
01303 
01304     /// <summary>
01305     /// Get the contents of a structured text character.
01306     /// </summary>
01307     /// <param name="character">The address of the character.</param>
01308     /// <param name="out_c">Unicode code point of the character.</param>
01309     /// <param name="out_color">An sRGB hex representation of the colour of the character.</param>
01310     /// <param name="out_origin_x">The x coordinate of the baseline origin of the
01311     character.</param>
01312     /// <param name="out_origin_y">The y coordinate of the baseline origin of the
01313     character.</param>
01314     /// <param name="out_size">The size in points of the character.</param>
01315     /// <param name="out_ll_x">The x coordinate of the lower left corner of the bounding
01316     quad.</param>
01317     /// <param name="out_ll_y">The y coordinate of the lower left corner of the bounding
01318     quad.</param>
01319     /// <param name="out_ul_x">The x coordinate of the upper left corner of the bounding
01320     quad.</param>
01321     /// <param name="out_ul_y">The y coordinate of the upper left corner of the bounding
01322     quad.</param>
01323     /// <param name="out_ur_x">The x coordinate of the upper right corner of the bounding
01324     quad.</param>
01325     /// <param name="out_ur_y">The y coordinate of the upper right corner of the bounding
01326     quad.</param>
01327     /// <param name="out_lr_x">The x coordinate of the lower right corner of the bounding
01328     quad.</param>
01329     /// <param name="out_lr_y">The y coordinate of the lower right corner of the bounding
01330     quad.</param>
01331     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01332     occurred.</returns>
01333     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01334     internal static extern int GetStructuredTextChar(IntPtr character, ref int out_c, ref int
01335     out_color, ref float out_origin_x, ref float out_origin_y, ref float out_size, ref float out_ll_x,
01336     ref float out_ll_y, ref float out_ul_x, ref float out_ul_y, ref float out_ur_x, ref float out_ur_y,
01337     ref float out_lr_x, ref float out_lr_y);
01338 
01339     /// <summary>
01340     /// Get an array of structured text characters from a structured text line.
01341     /// </summary>
01342     /// <param name="line">The structured text line from which the characters should be
01343     extracted.</param>
01344     /// <param name="out_chars">An array of pointers to the structured text characters.</param>
01345     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01346     occurred.</returns>
01347     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01348     internal static extern int GetStructuredTextChars(IntPtr line, IntPtr out_chars);
01349 
01350     /// <summary>
01351     /// Get the contents of a structured text line.

```

```

01326     /// </summary>
01327     /// <param name="line">The address of the line.</param>
01328     /// <param name="out_wmode">An integer equivalent to <see cref="MuPDFStructuredTextLine"/>
01329     /// representing the writing mode of the line.</param>
01330     /// <param name="out_x0">The left coordinate in page units of the bounding box of the
01331     /// line.</param>
01332     /// <param name="out_y0">The top coordinate in page units of the bounding box of the
01333     /// line.</param>
01334     /// <param name="out_x1">The right coordinate in page units of the bounding box of the
01335     /// line.</param>
01336     /// <param name="out_y1">The bottom coordinate in page units of the bounding box of the
01337     /// line.</param>
01338     /// <param name="out_x">The x component of the normalised direction of the baseline.</param>
01339     /// <param name="out_y">The y component of the normalised direction of the baseline.</param>
01340     /// <param name="out_char_count">The number of characters in the line.</param>
01341     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01342     /// occurred.</returns>
01343     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01344     internal static extern int GetStructuredTextLine(IntPtr line, ref int out_wmode, ref float
01345     out_x0, ref float out_y0, ref float out_x1, ref float out_y1, ref float out_x, ref float out_y, ref
01346     int out_char_count);
01347
01348     /// <summary>
01349     /// Get an array of structured text lines from a structured text block.
01350     /// </summary>
01351     /// <param name="block">The structured text block from which the lines should be
01352     /// extracted.</param>
01353     /// <param name="out_lines">An array of pointers to the structured text lines.</param>
01354     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01355     /// occurred.</returns>
01356     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01357     internal static extern int GetStructuredTextLines(IntPtr block, IntPtr out_lines);
01358
01359     /// <summary>
01360     /// Get the contents of a structured text block.
01361     /// </summary>
01362     /// <param name="block">The address of the block.</param>
01363     /// <param name="out_type">An integer equivalent to <see
01364     /// cref="MuPDFStructuredTextBlock.Types"/> representing the type of the block.</param>
01365     /// <param name="out_x0">The left coordinate in page units of the bounding box of the
01366     /// block.</param>
01367     /// <param name="out_y0">The top coordinate in page units of the bounding box of the
01368     /// block.</param>
01369     /// <param name="out_x1">The right coordinate in page units of the bounding box of the
01370     /// block.</param>
01371     /// <param name="out_y1">The bottom coordinate in page units of the bounding box of the
01372     /// block.</param>
01373     /// <param name="out_line_count">The number of lines in the block.</param>
01374     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01375     /// occurred.</returns>
01376     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01377     internal static extern int GetStructuredTextBlock(IntPtr block, ref int out_type, ref float
01378     out_x0, ref float out_y0, ref float out_x1, ref float out_y1, ref int out_line_count);
01379
01380     /// <summary>
01381     /// Get an array of structured text blocks from a structured text page.
01382     /// </summary>
01383     /// <param name="page">The structured text page from which the blocks should be
01384     /// extracted.</param>
01385     /// <param name="out_blocks">An array of pointers to the structured text blocks.</param>
01386     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01387     /// occurred.</returns>
01388     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01389     internal static extern int GetStructuredTextBlocks(IntPtr page, IntPtr out_blocks);
01390
01391     /// <summary>
01392     /// Get a structured text representation of a display list.
01393     /// </summary>
01394     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01395     /// <param name="list">The display list whose structured text representation is
01396     /// sought.</param>
01397     /// <param name="out_page">The address of the structured text page.</param>
01398     /// <param name="out_stext_block_count">The number of structured text blocks in the
01399     /// page.</param>
01400     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01401     /// occurred.</returns>
01402     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01403     internal static extern int GetStructuredTextPage(IntPtr ctx, IntPtr list, ref IntPtr out_page,
01404     ref int out_stext_block_count);
01405
01406     /// <summary>
01407     /// Delegate defining a callback function that is invoked by the unmanaged MuPDF library to
01408     /// indicate OCR progress.
01409     /// </summary>
01410     /// <param name="progress">The current progress, ranging from 0 to 100.</param>
01411     /// <returns>This function should return 0 to indicate that the OCR process should continue,
01412     /// or 1 to indicate that it should be stopped.</returns>

```

```

01388     internal delegate int ProgressCallback(int progress);
01389
01390     /// <summary>
01391     /// Get a structured text representation of a display list, using the Tesseract OCR engine.
01392     /// </summary>
01393     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01394     /// <param name="list">The display list whose structured text representation is
01395     /// sought.</param>
01396     /// <param name="out_page">The address of the structured text page.</param>
01397     /// <param name="out_stext_block_count">The number of structured text blocks in the
01398     /// page.</param>
01399     /// <param name="zoom">How much the specified region should be scaled when rendering. This
01400     /// determines the size in pixels of the image that is passed to Tesseract.</param>
01401     /// <param name="x0">The left coordinate in page units of the region of the display list that
01402     /// should be analysed.</param>
01403     /// <param name="y0">The top coordinate in page units of the region of the display list that
01404     /// should be analysed.</param>
01405     /// <param name="x1">The right coordinate in page units of the region of the display list that
01406     /// should be analysed.</param>
01407     /// <param name="y1">The bottom coordinate in page units of the region of the display list
01408     /// that should be analysed.</param>
01409     /// <param name="prefix">A string value that will be used as an argument for the <c>putenv</c>
01410     /// function. If this is <see langword="null"/>, the <c>putenv</c> function is not invoked. Usually used
01411     /// to set the value of the <c>TESSDATA_PREFIX</c> environment variable.</param>
01412     /// <param name="language">The name of the language model file to use for the OCR.</param>
01413     /// <param name="callback">A progress callback function. This function will be called with an
01414     /// integer parameter ranging from 0 to 100 to indicate OCR progress, and should return 0 to continue or
01415     /// 1 to abort the OCR process.</param>
01416     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01417     /// occurred.</returns>
01418     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01419     internal static extern int GetStructuredTextPageWithOCR(IntPtr ctx, IntPtr list, ref IntPtr
01420     out_page, ref int out_stext_block_count, float zoom, float x0, float y0, float x1, float y1, string
01421     prefix, string language, [MarshalAs(UnmanagedType.FunctionPtr)] ProgressCallback callback);
01422
01423     /// <summary>
01424     /// Free a native structured text page and its associated resources.
01425     /// </summary>
01426     /// <param name="ctx"></param>
01427     /// <param name="page"></param>
01428     /// <returns>An integer equivalent to <see cref="ExitCodes"/> detailing whether any errors
01429     /// occurred.</returns>
01430     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01431     internal static extern int DisposeStructuredTextPage(IntPtr ctx, IntPtr page);
01432
01433     /// <summary>
01434     /// Redirect the standard output and standard error to named pipes with the specified names.
01435     /// On Windows, these are actually named pipes; on Linux and macOS, these are Unix sockets (matching the
01436     /// behaviour of System.IO.Pipes). Note that this has side-effects.
01437     /// </summary>
01438     /// <param name="stdoutFD">When the method returns, this variable will contain the file
01439     /// descriptor corresponding to the "real" stdout.</param>
01440     /// <param name="stderrFD">When the method returns, this variable will contain the file
01441     /// descriptor corresponding to the "real" stderr.</param>
01442     /// <param name="stdoutPipeName">The name of the pipe where stdout will be redirected. On
01443     /// Windows, this should be of the form "\.\pipe\xxx", while on Linux and macOS it should be an absolute
01444     /// file path (maximum length 107/108 characters).</param>
01445     /// <param name="stderrPipeName">The name of the pipe where stderr will be redirected. On
01446     /// Windows, this should be of the form "\.\pipe\xxx", while on Linux and macOS it should be an absolute
01447     /// file path (maximum length 107/108 characters).</param>
01448     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01449     internal static extern void RedirectOutput(out int stdoutFD, out int stderrFD, string
01450     stdoutPipeName, string stderrPipeName);
01451
01452     /// <summary>
01453     /// Write the specified <paramref name="text"/> to a file descriptor. Use 1 for stdout and 2
01454     /// for stderr (which may have been redirected).
01455     /// </summary>
01456     /// <param name="fileDescriptor">The file descriptor on which to write.</param>
01457     /// <param name="text">The text to write.</param>
01458     /// <param name="length">The length of the text to write (i.e., text.Length).</param>
01459     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01460     internal static extern void WriteToFileDescriptor(int fileDescriptor, string text, int
01461     length);
01462
01463     /// <summary>
01464     /// Reset the standard output and standard error (or redirect them to the specified file
01465     /// descriptors, theoretically). Use with the <paramref name="stdoutFD"/> and <paramref name="stderrFD"/>
01466     /// returned by <see cref="RedirectOutput"/> to undo what it did.
01467     /// </summary>
01468     /// <param name="stdoutFD">The file descriptor corresponding to the "real" stdout.</param>
01469     /// <param name="stderrFD">The file descriptor corresponding to the "real" stderr.</param>
01470     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01471     internal static extern void ResetOutput(int stdoutFD, int stderrFD);
01472
01473     /// <summary>
01474     /// Unlocks a document with a password.
01475 
```

```

01447     /// </summary>
01448     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01449     /// <param name="doc">The document that needs to be unlocked.</param>
01450     /// <param name="password">The password to unlock the document.</param>
01451     /// <returns>0 if the document could not be unlocked, 1 if the document did not require
unlocking in the first place, 2 if the document was unlocked using the user password and 4 if the
document was unlocked using the owner password.</returns>
01452     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01453     internal static extern int UnlockWithPassword(IntPtr ctx, IntPtr doc, string password);
01454
01455     /// <summary>
01456     /// Checks whether a password is required to open the document.
01457     /// </summary>
01458     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01459     /// <param name="doc">The document that needs to be checked.</param>
01460     /// <returns>0 if a password is not needed, 1 if a password is needed.</returns>
01461     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01462     internal static extern int CheckIfPasswordNeeded(IntPtr ctx, IntPtr doc);
01463
01464     /// <summary>
01465     /// Returns the current permissions for the document. Note that these are not actually
enforced.
01466     /// </summary>
01467     /// <param name="ctx">A context to hold the exception stack and the cached resources.</param>
01468     /// <param name="doc">The document whose permissions need to be checked.</param>
01469     /// <returns>An integer with bit 0 set if the document can be printed, bit 1 set if it can be
copied, bit 2 set if it can be edited, and bit 3 set if it can be annotated.</returns>
01470     [DllImport("MuPDFWrapper", CallingConvention = CallingConvention.Cdecl)]
01471     internal static extern int GetPermissions(IntPtr ctx, IntPtr doc);
01472 }
01473 }
```

8.5 MuPDFContext.cs

```

00001 /*
00002  * MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
00003  * Copyright (C) 2020 Giorgio Bianchini
00004
00005  This program is free software: you can redistribute it and/or modify
00006  it under the terms of the GNU Affero General Public License as
00007  published by the Free Software Foundation, version 3.
00008
00009  This program is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU Affero General Public License for more details.
00013
00014  You should have received a copy of the GNU Affero General Public License
00015  along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using System;
00019
00020 namespace MuPDFCore
00021 {
00022     /// <summary>
00023     /// A wrapper around a MuPDF context object, which contains the exception stack and the resource
cache store.
00024     /// </summary>
00025     public class MuPDFContext : IDisposable
00026     {
00027         /// <summary>
00028         /// A pointer to the native context object.
00029         /// </summary>
00030         internal readonly IntPtr NativeContext;
00031
00032         /// <summary>
00033         /// The current size in bytes of the resource cache store. Read-only.
00034         /// </summary>
00035         public long StoreSize
00036         {
00037             get
00038             {
00039                 return (long)NativeMethods.GetCurrentStoreSize(this.NativeContext);
00040             }
00041         }
00042
00043         /// <summary>
00044         /// The maximum size in bytes of the resource cache store. Read-only.
00045         /// </summary>
00046         public long StoreMaxSize
00047         {
00048             get
00049         }
00050     }
00051 }
```

```

00049         {
00050             return (long)NativeMethods.GetMaxStoreSize(this.NativeContext);
00051         }
00052     }
00053
00054     /// <summary>
00055     /// Create a new <see cref="MuPDFContext"/> instance with the specified cache store size.
00056     /// </summary>
00057     /// <param name="storeSize">The maximum size in bytes of the resource cache store. The default
00058     /// value is 256 MiB.</param>
00059     public MuPDFContext(uint storeSize = 256 << 20)
00060     {
00061         ExitCodes result = (ExitCodes)NativeMethods.CreateContext((ulong)storeSize, ref
00062         NativeContext);
00063
00064         switch (result)
00065         {
00066             case ExitCodes.EXIT_SUCCESS:
00067                 break;
00068             case ExitCodes.ERR_CANNOT_CREATE_CONTEXT:
00069                 throw new MuPDFException("Cannot create MuPDF context", result);
00070             case ExitCodes.ERR_CANNOT_REGISTER_HANDLERS:
00071                 throw new MuPDFException("Cannot register document handlers", result);
00072             default:
00073                 throw new MuPDFException("Unknown error", result);
00074         }
00075
00076     /// <summary>
00077     /// Wrap an existing pointer to a native MuPDF context object.
00078     /// </summary>
00079     /// <param name="nativeContext">The pointer to the native context that should be used.</param>
00080     internal MuPDFContext(IntPtr nativeContext)
00081     {
00082         this.NativeContext = nativeContext;
00083     }
00084
00085     /// <summary>
00086     /// Evict all items from the resource cache store (freeing the memory where they were held).
00087     /// </summary>
00088     public void ClearStore()
00089     {
00090         NativeMethods.EmptyStore(this.NativeContext);
00091     }
00092
00093     /// <summary>
00094     /// Evict items from the resource cache store (freeing the memory where they were held) until
00095     /// the size of the store drops to the specified fraction of the current size.
00096     /// </summary>
00097     /// <param name="fraction">The fraction of the current size that constitutes the target size
00098     /// of the store. If this is < 0, the cache is cleared. If this is >= 1, nothing happens.</param>
00099     public void ShrinkStore(double fraction)
00100     {
00101         if (fraction <= 0)
00102         {
00103             ClearStore();
00104         }
00105         else if (Math.Round(fraction * 100) < 100)
00106         {
00107             NativeMethods.ShrinkStore(this.NativeContext, (uint)Math.Round(fraction * 100));
00108         }
00109
00110     private bool disposedValue;
00111
00112     ///<inheritDoc/>
00113     protected virtual void Dispose(bool disposing)
00114     {
00115         if (!disposedValue)
00116         {
00117             if (disposing)
00118             {
00119                 // TODO: eliminare lo stato gestito (oggetti gestiti)
00120
00121                 NativeMethods.DisposeContext(NativeContext);
00122                 disposedValue = true;
00123             }
00124
00125     ///<inheritDoc/>
00126     ~MuPDFContext()
00127     {
00128         Dispose(disposing: false);
00129     }
00130
00131     ///<inheritDoc/>

```

```

00132     public void Dispose()
00133     {
00134         Dispose(disposing: true);
00135         GC.SuppressFinalize(this);
00136     }
00137 }
00138 }
```

8.6 MuPDFDisplayList.cs

```

00001 /*
00002  * MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
00003  * Copyright (C) 2020 Giorgio Bianchini
00004
00005  * This program is free software: you can redistribute it and/or modify
00006  * it under the terms of the GNU Affero General Public License as
00007  * published by the Free Software Foundation, version 3.
00008
00009  * This program is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU Affero General Public License for more details.
00013
00014  * You should have received a copy of the GNU Affero General Public License
00015  * along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using System;
00019
00020 namespace MuPDFCore
00021 {
00022     /// <summary>
00023     /// A wrapper around a MuPDF display list object, which contains the necessary informations to
00024     /// render a page to an image.
00025     internal class MuPDFDisplayList : IDisposable
00026     {
00027         /// <summary>
00028         /// The context that owns the document that was used to create this display list.
00029         /// </summary>
00030         private readonly MuPDFContext OwnerContext;
00031
00032         /// <summary>
00033         /// A pointer to the native display list object.
00034         /// </summary>
00035         readonly internal IntPtr NativeDisplayList;
00036
00037         /// <summary>
00038         /// The display list's bounds in page units. Read-only.
00039         /// </summary>
00040         public Rectangle Bounds { get; }
00041
00042         /// <summary>
00043         /// Create a new <see cref="MuPDFDisplayList"/> instance from the specified page.
00044         /// </summary>
00045         /// <param name="context">The context that owns the document from which the page was
00046         /// taken.</param>
00047         /// <param name="page">The page from which the display list should be generated.</param>
00048         /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00049         /// signatures) are included in the display list that is generated. Otherwise, only the page contents are
00050         /// included.</param>
00051         public MuPDFDisplayList(MuPDFContext context, MuPDFPage page, bool includeAnnotations = true)
00052         {
00053             this.OwnerContext = context;
00054
00055             float x0 = 0;
00056             float y0 = 0;
00057             float x1 = 0;
00058             float y1 = 0;
00059
00060             ExitCodes result = (ExitCodes)NativeMethods.GetDisplayList(context.NativeContext,
00061             page.NativePage, includeAnnotations ? 1 : 0, ref NativeDisplayList, ref x0, ref y0, ref x1, ref y1);
00062
00063             switch (result)
00064             {
00065                 case ExitCodes.EXIT_SUCCESS:
00066                     break;
00067                 case ExitCodes.ERR_CANNOT_RENDER:
00068                     throw new MuPDFException("Cannot render page", result);
00069                 default:
00070                     throw new MuPDFException("Unknown error", result);
00071             }
00072         }
00073     }
```

```

00069         this.Bounds = new Rectangle(Math.Round(x0 * page.OwnerDocument.ImageXRes / 72.0 * 1000) /
00070             1000, Math.Round(y0 * page.OwnerDocument.ImageYRes / 72.0 * 1000) / 1000, Math.Round(x1 *
00071             page.OwnerDocument.ImageXRes / 72.0 * 1000) / 1000, Math.Round(y1 * page.OwnerDocument.ImageYRes /
00072             72.0 * 1000) / 1000);
00073     }
00074     private bool disposedValue;
00075     protected virtual void Dispose(bool disposing)
00076     {
00077         if (!disposedValue)
00078         {
00079             NativeMethods.DisposeDisplayList(OwnerContext.NativeContext, NativeDisplayList);
00080             disposedValue = true;
00081         }
00082     }
00083     ~MuPDFDisplayList()
00084     {
00085         Dispose(disposing: false);
00086     }
00087     public void Dispose()
00088     {
00089         Dispose(disposing: true);
00090         GC.SuppressFinalize(this);
00091     }
00092 }
00093 }
```

8.7 MuPDFDocument.cs

```

00001 /*
00002     MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
00003     Copyright (C) 2020 Giorgio Bianchini
00004
00005     This program is free software: you can redistribute it and/or modify
00006     it under the terms of the GNU Affero General Public License as
00007     published by the Free Software Foundation, version 3.
00008
00009     This program is distributed in the hope that it will be useful,
00010     but WITHOUT ANY WARRANTY; without even the implied warranty of
00011     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012     GNU Affero General Public License for more details.
00013
00014     You should have received a copy of the GNU Affero General Public License
00015     along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using System;
00019 using System.IO;
00020 using System.Runtime.InteropServices;
00021 using System.Text;
00022 using System.Threading;
00023 using System.Threading.Tasks;
00024
00025 namespace MuPDFCore
00026 {
00027     /// <summary>
00028     /// A wrapper over a MuPDF document object, which contains possibly multiple pages.
00029     /// </summary>
00030     public class MuPDFDocument : IDisposable
00031     {
00032         /// <summary>
00033         /// If the document is an image, the horizontal resolution of the image. Otherwise, 72.
00034         /// </summary>
00035         internal double ImageXRes = double.NaN;
00036
00037         /// <summary>
00038         /// If the document is an image, the vertical resolution of the image. Otherwise, 72.
00039         /// </summary>
00040         internal double ImageYRes = double.NaN;
00041
00042         /// <summary>
00043         /// File extensions corresponding to the supported input formats.
00044         /// </summary>
00045         private static readonly string[] FileTypeMagics = new[]
00046         {
00047             ".pdf",
00048             ".xps",
00049             ".cbz",
00050             ".png",
00051             ".jpg",
00052         };
00053     }
00054 }
```

```
00052         ".bmp",
00053         ".gif",
00054         ".tif",
00055         ".pnm",
00056         ".pam",
00057         ".epub",
00058         ".fb2"
00059     };
00060
00061     /// <summary>
00062     /// An <see cref="IDisposable"/> with a value of null.
00063     /// </summary>
00064     private static IDisposable NullDataHolder = null;
00065
00066     /// <summary>
00067     /// The context that owns this document.
00068     /// </summary>
00069     private readonly MuPDFContext OwnerContext;
00070
00071     /// <summary>
00072     /// A pointer to the native document object.
00073     /// </summary>
00074     internal readonly IntPtr NativeDocument;
00075
00076     /// <summary>
00077     /// A pointer to the native stream that was used to create this document (if any).
00078     /// </summary>
00079     private readonly IntPtr NativeStream = IntPtr.Zero;
00080
00081     /// <summary>
00082     /// The number of pages in the document.
00083     /// </summary>
00084     private readonly int PageCount;
00085
00086     /// <summary>
00087     /// An <see cref="IDisposable"/> that will be disposed together with this object.
00088     /// </summary>
00089     private readonly IDisposable DataHolder = null;
00090
00091     /// <summary>
00092     /// A <see cref="GCHandle"/> that will be freed when this object is disposed.
00093     /// </summary>
00094     private GCHandle? DataHandle = null;
00095
00096     /// <summary>
00097     /// An array of <see cref="MuPDFDisplayList"/>, one for each page in the document.
00098     /// </summary>
00099     private readonly MuPDFDisplayList[] DisplayLists;
00100
00101     /// <summary>
00102     /// The pages contained in the document.
00103     /// </summary>
00104     public MuPDFPageCollection Pages { get; }
00105
00106     /// <summary>
00107     /// Defines whether the images resulting from rendering operations should be clipped to the
00108     /// page boundaries.
00109     /// </summary>
00110     public bool ClipToPageBounds { get; set; } = true;
00111
00112     /// <summary>
00113     /// Describes the encryption state of the document.
00114     /// </summary>
00115     public EncryptionState EncryptionState { get; private set; }
00116
00117     /// <summary>
00118     /// Describes the restriction state of the document.
00119     /// </summary>
00120     public RestrictionState RestrictionState { get; private set; }
00121
00122     /// <summary>
00123     /// Describes the operations that are restricted on the document. This is not actually
00124     /// enforced by the library,
00125     /// but library users should only allow these operations if the document has been unlocked
00126     /// with the owner password
00127     /// (i.e. if <see cref="RestrictionState"/> is <see cref="RestrictionState.Unlocked"/>).
00128     /// </summary>
00129     public DocumentRestrictions Restrictions { get; private set; }
00130
00131     /// <summary>
00132     /// Create a new <see cref="MuPDFDocument"/> from data bytes accessible through the specified
00133     /// pointer.
00134     /// </summary>
00135     /// <param name="context">The context that will own this document.</param>
00136     /// <param name="dataAddress">A pointer to the data bytes that make up the document.</param>
00137     /// <param name="dataLength">The number of bytes to read from the specified address.</param>
00138     /// <param name="fileType">The type of the document to read.</param>
```

```

00135     public MuPDFDocument(MuPDFContext context, IntPtr dataAddress, long dataLength, InputFileTypes
00136     fileType) : this(context, dataAddress, dataLength, fileType, ref NullDataHolder) { }
00137 
00138     /// <summary>
00139     /// Create a new <see cref="MuPDFDocument"/> from data bytes accessible through the specified
00140     /// pointer.
00141     /// </summary>
00142     /// <param name="context">The context that will own this document.</param>
00143     /// <param name="dataAddress">A pointer to the data bytes that make up the document.</param>
00144     /// <param name="dataLength">The number of bytes to read from the specified address.</param>
00145     /// <param name="fileType">The type of the document to read.</param>
00146     /// <param name="dataHolder">An <see cref="IDisposable"/> that will be disposed when the <see
00147     /// cref="MuPDFDocument"/> is disposed.</param>
00148     public MuPDFDocument(MuPDFContext context, IntPtr dataAddress, long dataLength, InputFileTypes
00149     fileType, ref IDisposable dataHolder)
00150     {
00151         bool isImage = fileType == InputFileTypes.BMP || fileType == InputFileTypes.GIF ||
00152         fileType == InputFileTypes.JPG || fileType == InputFileTypes.PAM || fileType == InputFileTypes.PNG
00153         || fileType == InputFileTypes.PNG || fileType == InputFileTypes.TIFF;
00154 
00155         this.OwnerContext = context;
00156         float xRes = 0;
00157         float yRes = 0;
00158 
00159         ExitCodes result =
00160         (ExitCodes)NativeMethods.CreateDocumentFromStream(context.NativeContext, dataAddress,
00161         (ulong)dataLength, FileTypeMagics[(int)fileType], isImage ? 1 : 0, ref NativeDocument, ref
00162         NativeStream, ref PageCount, ref xRes, ref yRes);
00163 
00164         if (xRes > 72)
00165         {
00166             this.ImageXRes = xRes;
00167         }
00168         else
00169         {
00170             this.ImageXRes = 72;
00171         }
00172 
00173         if (yRes > 72)
00174         {
00175             this.ImageYRes = yRes;
00176         }
00177         else
00178         {
00179             this.ImageYRes = 72;
00180         }
00181 
00182         this.DataHolder = dataHolder;
00183 
00184         switch (result)
00185         {
00186             case ExitCodes.EXIT_SUCCESS:
00187                 break;
00188             case ExitCodes.ERR_CANNOT_OPEN_STREAM:
00189                 throw new MuPDFException("Cannot open data stream", result);
00190             case ExitCodes.ERR_CANNOT_OPEN_FILE:
00191                 throw new MuPDFException("Cannot open document", result);
00192             case ExitCodes.ERR_CANNOT_COUNT_PAGES:
00193                 throw new MuPDFException("Cannot count pages", result);
00194             default:
00195                 throw new MuPDFException("Unknown error", result);
00196         }
00197 
00198         int permissions = NativeMethods.GetPermissions(context.NativeContext,
00199         this.NativeDocument);
00200 
00201         int restrictions = 0;
00202 
00203         if ((permissions & 1) == 0)
00204         {
00205             restrictions |= 1;
00206         }
00207 
00208         if ((permissions & 2) == 0)
00209         {
00210             restrictions |= 2;
00211         }

```

```

00212
00213     if ((permissions & 4) == 0)
00214     {
00215         restrictions |= 4;
00216     }
00217
00218     if ((permissions & 8) == 0)
00219     {
00220         restrictions |= 8;
00221     }
00222
00223     if (restrictions == 0)
00224     {
00225         this.Restrictions = DocumentRestrictions.None;
00226         this.RestrictionState = RestrictionState.Unrestricted;
00227     }
00228     else
00229     {
00230         this.Restrictions = (DocumentRestrictions)restrictions;
00231         this.RestrictionState = RestrictionState.Restricted;
00232     }
00233
00234     Pages = new MuPDFPageCollection(context, this, PageCount);
00235     DisplayLists = new MuPDFDisplayList[PageCount];
00236 }
00237
00238 /// <summary>
00239 /// Create a new <see cref="MuPDFDocument"/> from an array of bytes.
00240 /// </summary>
00241 /// <param name="context">The context that will own this document.</param>
00242 /// <param name="data">An array containing the data bytes that make up the document. This must
00243 not be altered until after the <see cref="MuPDFDocument"/> has been disposed!
00244     /// The address of the array will be pinned, which may cause degradation in the Garbage
00245     /// Collector's performance, and is thus only advised for short-lived documents. To avoid this issue,
00246     /// marshal the bytes to unmanaged memory and use one of the <see cref="IntPtr"/> constructors.</param>
00247     /// <param name="fileType">The type of the document to read.</param>
00248     public MuPDFDocument(MuPDFContext context, byte[] data, InputFileTypes fileType)
00249     {
00250         bool isImage = fileType == InputFileTypes.BMP || fileType == InputFileTypes.GIF ||
00251         fileType == InputFileTypes.JPG || fileType == InputFileTypes.PAM || fileType == InputFileTypes.PNG
00252         || fileType == InputFileTypes.PNM || fileType == InputFileTypes.TIFF;
00253
00254         this.OwnerContext = context;
00255
00256         DataHandle = GCHandle.Alloc(data, GCHandleType.Pinned);
00257         IntPtr dataAddress = DataHandle.Value.AddrOfPinnedObject();
00258         ulong dataLength = (ulong)data.Length;
00259
00260         float xRes = 0;
00261         float yRes = 0;
00262
00263         ExitCodes result =
00264             (ExitCodes)NativeMethods.CreateDocumentFromStream(context.NativeContext, dataAddress, dataLength,
00265             FileTypeMagics[(int)fileType], isImage ? 1 : 0, ref NativeDocument, ref NativeStream, ref PageCount,
00266             ref xRes, ref yRes);
00267
00268         if (xRes > 72)
00269         {
00270             this.ImageXRes = xRes;
00271         }
00272         else
00273         {
00274             this.ImageXRes = 72;
00275         }
00276
00277         if (yRes > 72)
00278         {
00279             this.ImageYRes = yRes;
00280         }
00281         else
00282         {
00283             this.ImageYRes = 72;
00284         }
00285
00286         switch (result)
00287         {
00288             case ExitCodes.EXIT_SUCCESS:
00289                 break;
00290             case ExitCodes.ERR_CANNOT_OPEN_STREAM:
00291                 throw new MuPDFException("Cannot open data stream", result);
00292             case ExitCodes.ERR_CANNOT_OPEN_FILE:
00293                 throw new MuPDFException("Cannot open document", result);
00294             case ExitCodes.ERR_CANNOT_COUNT_PAGES:
00295                 throw new MuPDFException("Cannot count pages", result);
00296             default:
00297                 throw new MuPDFException("Unknown error", result);
00298         }
00299     }

```

```

00291             if (NativeMethods.CheckIfPasswordNeeded(context.NativeContext, this.NativeDocument) != 0)
00292             {
00293                 this.EncryptionState = EncryptionState.Encrypted;
00294             }
00295             else
00296             {
00297                 this.EncryptionState = EncryptionState.Unencrypted;
00298             }
00299         }
00300     }
00301     int permissions = NativeMethods.GetPermissions(context.NativeContext,
00302         this.NativeDocument);
00302     int restrictions = 0;
00303     if ((permissions & 1) == 0)
00304     {
00305         restrictions |= 1;
00306     }
00307     if ((permissions & 2) == 0)
00308     {
00309         restrictions |= 2;
00310     }
00311     if ((permissions & 4) == 0)
00312     {
00313         restrictions |= 4;
00314     }
00315     if ((permissions & 8) == 0)
00316     {
00317         restrictions |= 8;
00318     }
00319     if (restrictions == 0)
00320     {
00321         this.Restrictions = DocumentRestrictions.None;
00322         this.RestrictionState = RestrictionState.Unrestricted;
00323     }
00324     else
00325     {
00326         this.Restrictions = (DocumentRestrictions)restrictions;
00327         this.RestrictionState = RestrictionState.Restricted;
00328     }
00329     Pages = new MuPDFPageCollection(context, this, PageCount);
00330     DisplayLists = new MuPDFDisplayList[PageCount];
00331 }
00332 }
00333 }
00334 }
00335 }
00336 }
00337 }
00338 }
00339 }
00340 }
00341 }
00342 }
00343 }
00344 }
00345 }
00346 }
00347 }
00348 }
00349 }
00350 }
00351 }
00352 }
00353 }
00354 }
00355 }
00356 }
00357 }
00358 }
00359 }
00360 }
00361 }
00362 }
00363 }
00364 }
00365 }
00366 }

    if (NativeMethods.CreateDocumentFromStream(context.NativeContext, dataAddress, dataLength,
        fileTypeMagics[(int)fileType], isImage ? 1 : 0, ref NativeDocument, ref NativeStream, ref PageCount,
        ref xRes, ref yRes);

```

```
00367         if (xRes > 72)
00368         {
00369             this.ImageXRes = xRes;
00370         }
00371     else
00372     {
00373         this.ImageXRes = 72;
00374     }
00375
00376     if (yRes > 72)
00377     {
00378         this.ImageYRes = yRes;
00379     }
00380     else
00381     {
00382         this.ImageYRes = 72;
00383     }
00384
00385
00386     switch (result)
00387     {
00388         case ExitCodes.EXIT_SUCCESS:
00389             break;
00390         case ExitCodes.ERR_CANNOT_OPEN_STREAM:
00391             throw new MuPDFException("Cannot open data stream", result);
00392         case ExitCodes.ERR_CANNOT_OPEN_FILE:
00393             throw new MuPDFException("Cannot open document", result);
00394         case ExitCodes.ERR_CANNOT_COUNT_PAGES:
00395             throw new MuPDFException("Cannot count pages", result);
00396         default:
00397             throw new MuPDFException("Unknown error", result);
00398     }
00399
00400     if (NativeMethods.CheckIfPasswordNeeded(context.NativeContext, this.NativeDocument) != 0)
00401     {
00402         this.EncryptionState = EncryptionState.Encrypted;
00403     }
00404     else
00405     {
00406         this.EncryptionState = EncryptionState.Unencrypted;
00407     }
00408
00409     int permissions = NativeMethods.GetPermissions(context.NativeContext,
this.NativeDocument);
00410
00411     int restrictions = 0;
00412
00413     if ((permissions & 1) == 0)
00414     {
00415         restrictions |= 1;
00416     }
00417
00418     if ((permissions & 2) == 0)
00419     {
00420         restrictions |= 2;
00421     }
00422
00423     if ((permissions & 4) == 0)
00424     {
00425         restrictions |= 4;
00426     }
00427
00428     if ((permissions & 8) == 0)
00429     {
00430         restrictions |= 8;
00431     }
00432
00433     if (restrictions == 0)
00434     {
00435         this.Restrictions = DocumentRestrictions.None;
00436         this.RestrictionState = RestrictionState.Unrestricted;
00437     }
00438     else
00439     {
00440         this.Restrictions = (DocumentRestrictions)restrictions;
00441         this.RestrictionState = RestrictionState.Restricted;
00442     }
00443
00444     Pages = new MuPDFPageCollection(context, this, PageCount);
00445     DisplayLists = new MuPDFDisplayList[PageCount];
00446 }
00447
00448 /// <summary>
00449 /// Create a new <see cref="MuPDFDocument"/> from a file.
00450 /// </summary>
00451 /// <param name="context">The context that will own this document.</param>
00452 /// <param name="fileName">The path to the file to open.</param>
```

```

00453     public MuPDFDocument(MuPDFContext context, string fileName)
00454     {
00455         bool isImage;
00456
00457         string extension = Path.GetExtension(fileName).ToLowerInvariant();
00458
00459         switch (extension)
00460     {
00461             case ".bmp":
00462             case ".dib":
00463
00464             case ".gif":
00465
00466             case ".jpg":
00467             case ".jpeg":
00468             case ".jpe":
00469             case ".jfif":
00470             case ".jfi":
00471
00472             case ".pam":
00473             case ".pbm":
00474             case ".pgm":
00475             case ".ppm":
00476             case ".pnm":
00477
00478             case ".png":
00479
00480             case ".tif":
00481             case ".tiff":
00482                 isImage = true;
00483                 break;
00484             default:
00485                 isImage = false;
00486                 break;
00487         }
00488
00489
00490         this.OwnerContext = context;
00491
00492         float xRes = 0;
00493         float yRes = 0;
00494
00495         ExitCodes result;
00496
00497         using (UTF8EncodedString encodedFileName = new UTF8EncodedString(fileName))
00498     {
00499             result = (ExitCodes)NativeMethods.CreateDocumentFromFile(context.NativeContext,
00500             encodedFileName.Address, isImage ? 1 : 0, ref NativeDocument, ref PageCount, ref xRes, ref yRes);
00501
00502
00503             if (xRes > 72)
00504         {
00505                 this.ImageXRes = xRes;
00506             }
00507             else
00508         {
00509                 this.ImageXRes = 72;
00510             }
00511
00512             if (yRes > 72)
00513         {
00514                 this.ImageYRes = yRes;
00515             }
00516             else
00517         {
00518                 this.ImageYRes = 72;
00519             }
00520
00521             switch (result)
00522         {
00523             case ExitCodes.EXIT_SUCCESS:
00524                 break;
00525             case ExitCodes.ERR_CANNOT_OPEN_FILE:
00526                 throw new MuPDFException("Cannot open document", result);
00527             case ExitCodes.ERR_CANNOT_COUNT_PAGES:
00528                 throw new MuPDFException("Cannot count pages", result);
00529             default:
00530                 throw new MuPDFException("Unknown error", result);
00531         }
00532
00533             if (NativeMethods.CheckIfPasswordNeeded(context.NativeContext, this.NativeDocument) != 0)
00534         {
00535                 this.EncryptionState = EncryptionState.Encrypted;
00536             }
00537             else
00538         {

```

```

00539         this.EncryptionState = EncryptionState.Unencrypted;
00540     }
00541 
00542     int permissions = NativeMethods.GetPermissions(context.NativeContext,
00543         this.NativeDocument);
00544 
00545     int restrictions = 0;
00546 
00547     if ((permissions & 1) == 0)
00548     {
00549         restrictions |= 1;
00550     }
00551 
00552     if ((permissions & 2) == 0)
00553     {
00554         restrictions |= 2;
00555     }
00556 
00557     if ((permissions & 4) == 0)
00558     {
00559         restrictions |= 4;
00560     }
00561 
00562     if ((permissions & 8) == 0)
00563     {
00564         restrictions |= 8;
00565     }
00566 
00567     if (restrictions == 0)
00568     {
00569         this.Restrictions = DocumentRestrictions.None;
00570         this.RestrictionState = RestrictionState.Unrestricted;
00571     }
00572     else
00573     {
00574         this.Restrictions = (DocumentRestrictions)restrictions;
00575         this.RestrictionState = RestrictionState.Restricted;
00576     }
00577 
00578     Pages = new MuPDFPageCollection(context, this, PageCount);
00579     DisplayLists = new MuPDFDisplayList[PageCount];
00580 }
00581 
00582 /// <summary>
00583 /// Discard all the display lists that have been loaded from the document, possibly freeing
00584 /// some memory in the case of a huge document.
00585 /// </summary>
00586 public void ClearCache()
00587 {
00588     for (int i = 0; i < PageCount; i++)
00589     {
00590         DisplayLists[i]?.Dispose();
00591         DisplayLists[i] = null;
00592     }
00593 
00594 /// <summary>
00595 /// Render (part of) a page to an array of bytes.
00596 /// </summary>
00597 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00598 /// <param name="region">The region of the page to render in page units.</param>
00599 /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00600 /// size in pixel of the image.</param>
00601 /// <param name="pixelFormat">The format of the pixel data.</param>
00602 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00603 /// signatures) are included in the display list that is generated. Otherwise, only the page contents are
00604 /// included.</param>
00605 /// <returns>A byte array containing the raw values for the pixels of the rendered
00606 /// image.</returns>
00607 public byte[] Render(int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat,
00608     bool includeAnnotations = true)
00609     {
00610         if (this.EncryptionState == EncryptionState.Encrypted)
00611         {
00612             throw new DocumentLockedException("A password is necessary to render the document!");
00613         }
00614 
00615         int bufferSize = MuPDFDocument.GetRenderedSize(region, zoom, pixelFormat);
00616 
00617         byte[] buffer = new byte[bufferSize];
00618 
00619         GCHandle bufferHandle = GCHandle.Alloc(buffer, GCHandleType.Pinned);
00620         IntPtr bufferPointer = bufferHandle.AddrOfPinnedObject();
00621 
00622         try
00623         {
00624             Render(pageNumber, region, zoom, pixelFormat, bufferPointer, includeAnnotations);
00625         }
00626     }

```

```

00619         }
00620         finally
00621     {
00622         bufferHandle.Free();
00623     }
00624
00625     return buffer;
00626 }
00627
00628 /// <summary>
00629 /// Render a page to an array of bytes.
00630 /// </summary>
00631 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00632 /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00633 size in pixel of the image.</param>
00634 /// <param name="pixelFormat">The format of the pixel data.</param>
00635 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00636 signatures) are included in the display list that is generated. Otherwise, only the page contents are
00637 included.</param>
00638 /// <returns>A byte array containing the raw values for the pixels of the rendered
00639 image.</returns>
00640 public byte[] Render(int pageNumber, double zoom, PixelFormats pixelFormat, bool
00641 includeAnnotations = true)
00642 {
00643     if (this.EncryptionState == EncryptionState.Encrypted)
00644     {
00645         throw new DocumentLockedException("A password is necessary to render the document!");
00646     }
00647
00648     Rectangle region = this.Pages[pageNumber].Bounds;
00649     return Render(pageNumber, region, zoom, pixelFormat, includeAnnotations);
00650 }
00651
00652 /// <summary>
00653 /// Render (part of) a page to the specified destination.
00654 /// </summary>
00655 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00656 /// <param name="region">The region of the page to render in page units.</param>
00657 /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00658 size in pixel of the image.</param>
00659 /// <param name="pixelFormat">The format of the pixel data.</param>
00660 /// <param name="destination">The address of the buffer where the pixel data will be written.
00661 There must be enough space available to write the values for all the pixels, otherwise this will fail
00662 catastrophically!</param>
00663 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00664 signatures) are included in the display list that is generated. Otherwise, only the page contents are
00665 included.</param>
00666     public void Render(int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat,
00667 IntPtr destination, bool includeAnnotations = true)
00668     {
00669         if (this.EncryptionState == EncryptionState.Encrypted)
00670         {
00671             throw new DocumentLockedException("A password is necessary to render the document!");
00672         }
00673
00674         if (DisplayLists[pageNumber] == null)
00675         {
00676             DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
00677 this.Pages[pageNumber], includeAnnotations);
00678         }
00679
00680         if (zoom < 0.000001 || zoom * region.Width <= 0.001 || zoom * region.Height <= 0.001)
00681         {
00682             throw new ArgumentOutOfRangeException(nameof(zoom), zoom, "The zoom factor is too
00683 small!");
00684         }
00685
00686         if (this.ImageXRes != 72 || this.ImageYRes != 72)
00687         {
00688             zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
00689             region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 /
00690 this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
00691         }
00692
00693         float fzoom = (float)zoom;
00694
00695         ExitCodes result =
00696         (ExitCodes)NativeMethods.RenderSubDisplayList(OwnerContext.NativeContext,
00697 DisplayLists[pageNumber].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, fzoom,
00698 (int)pixelFormat, destination, IntPtr.Zero);
00699
00700         switch (result)
00701     {
00702         case ExitCodes.EXIT_SUCCESS:
00703             break;
00704         case ExitCodes.ERR_CANNOT_RENDER:
00705             throw new MuPDFException("Cannot render page", result);
00706     }
00707 }
00708
00709 /// <summary>
00710 /// Get the page count of the document.
00711 /// </summary>
00712 /// <returns>The number of pages in the document.
00713 /// </returns>
00714 public int PageCount { get; }
00715
00716 /// <summary>
00717 /// Get the page at the specified index.
00718 /// </summary>
00719 /// <param name="index">The index of the page to get.
00720 /// </param>
00721 /// <returns>The page at the specified index.
00722 /// </returns>
00723 public Page this[int index]
00724 {
00725     get { return Pages[index]; }
00726 }
00727
00728 /// <summary>
00729 /// Get the page at the specified index.
00730 /// </summary>
00731 /// <param name="index">The index of the page to get.
00732 /// </param>
00733 /// <param name="page">The page to get.
00734 /// </param>
00735 /// </returns>
00736 public void SetPage(int index, Page page)
00737 {
00738     Pages[index] = page;
00739 }
00740
00741 /// <summary>
00742 /// Get the page at the specified index.
00743 /// </summary>
00744 /// <param name="index">The index of the page to get.
00745 /// </param>
00746 /// <param name="page">The page to get.
00747 /// </param>
00748 /// </returns>
00749 public Page GetPage(int index)
00750 {
00751     return Pages[index];
00752 }
00753
00754 /// <summary>
00755 /// Get the page at the specified index.
00756 /// </summary>
00757 /// <param name="index">The index of the page to get.
00758 /// </param>
00759 /// <param name="page">The page to get.
00760 /// </param>
00761 /// </returns>
00762 public void SetPage(int index, Page page)
00763 {
00764     Pages[index] = page;
00765 }
00766
00767 /// <summary>
00768 /// Get the page at the specified index.
00769 /// </summary>
00770 /// <param name="index">The index of the page to get.
00771 /// </param>
00772 /// <param name="page">The page to get.
00773 /// </param>
00774 /// </returns>
00775 public Page GetPage(int index)
00776 {
00777     return Pages[index];
00778 }
00779
00780 /// <summary>
00781 /// Get the page at the specified index.
00782 /// </summary>
00783 /// <param name="index">The index of the page to get.
00784 /// </param>
00785 /// <param name="page">The page to get.
00786 /// </param>
00787 /// </returns>
00788 public void SetPage(int index, Page page)
00789 {
00790     Pages[index] = page;
00791 }
00792
00793 /// <summary>
00794 /// Get the page at the specified index.
00795 /// </summary>
00796 /// <param name="index">The index of the page to get.
00797 /// </param>
00798 /// <param name="page">The page to get.
00799 /// </param>
00800 /// </returns>
00801 public Page GetPage(int index)
00802 {
00803     return Pages[index];
00804 }
00805
00806 /// <summary>
00807 /// Get the page at the specified index.
00808 /// </summary>
00809 /// <param name="index">The index of the page to get.
00810 /// </param>
00811 /// <param name="page">The page to get.
00812 /// </param>
00813 /// </returns>
00814 public void SetPage(int index, Page page)
00815 {
00816     Pages[index] = page;
00817 }
00818
00819 /// <summary>
00820 /// Get the page at the specified index.
00821 /// </summary>
00822 /// <param name="index">The index of the page to get.
00823 /// </param>
00824 /// <param name="page">The page to get.
00825 /// </param>
00826 /// </returns>
00827 public Page GetPage(int index)
00828 {
00829     return Pages[index];
00830 }
00831
00832 /// <summary>
00833 /// Get the page at the specified index.
00834 /// </summary>
00835 /// <param name="index">The index of the page to get.
00836 /// </param>
00837 /// <param name="page">The page to get.
00838 /// </param>
00839 /// </returns>
00840 public void SetPage(int index, Page page)
00841 {
00842     Pages[index] = page;
00843 }
00844
00845 /// <summary>
00846 /// Get the page at the specified index.
00847 /// </summary>
00848 /// <param name="index">The index of the page to get.
00849 /// </param>
00850 /// <param name="page">The page to get.
00851 /// </param>
00852 /// </returns>
00853 public Page GetPage(int index)
00854 {
00855     return Pages[index];
00856 }
00857
00858 /// <summary>
00859 /// Get the page at the specified index.
00860 /// </summary>
00861 /// <param name="index">The index of the page to get.
00862 /// </param>
00863 /// <param name="page">The page to get.
00864 /// </param>
00865 /// </returns>
00866 public void SetPage(int index, Page page)
00867 {
00868     Pages[index] = page;
00869 }
00870
00871 /// <summary>
00872 /// Get the page at the specified index.
00873 /// </summary>
00874 /// <param name="index">The index of the page to get.
00875 /// </param>
00876 /// <param name="page">The page to get.
00877 /// </param>
00878 /// </returns>
00879 public Page GetPage(int index)
00880 {
00881     return Pages[index];
00882 }
00883
00884 /// <summary>
00885 /// Get the page at the specified index.
00886 /// </summary>
00887 /// <param name="index">The index of the page to get.
00888 /// </param>
00889 /// <param name="page">The page to get.
00890 /// </param>
00891 /// </returns>
00892 public void SetPage(int index, Page page)
00893 {
00894     Pages[index] = page;
00895 }
00896
00897 /// <summary>
00898 /// Get the page at the specified index.
00899 /// </summary>
00900 /// <param name="index">The index of the page to get.
00901 /// </param>
00902 /// <param name="page">The page to get.
00903 /// </param>
00904 /// </returns>
00905 public Page GetPage(int index)
00906 {
00907     return Pages[index];
00908 }
00909
00910 /// <summary>
00911 /// Get the page at the specified index.
00912 /// </summary>
00913 /// <param name="index">The index of the page to get.
00914 /// </param>
00915 /// <param name="page">The page to get.
00916 /// </param>
00917 /// </returns>
00918 public void SetPage(int index, Page page)
00919 {
00920     Pages[index] = page;
00921 }
00922
00923 /// <summary>
00924 /// Get the page at the specified index.
00925 /// </summary>
00926 /// <param name="index">The index of the page to get.
00927 /// </param>
00928 /// <param name="page">The page to get.
00929 /// </param>
00930 /// </returns>
00931 public Page GetPage(int index)
00932 {
00933     return Pages[index];
00934 }
00935
00936 /// <summary>
00937 /// Get the page at the specified index.
00938 /// </summary>
00939 /// <param name="index">The index of the page to get.
00940 /// </param>
00941 /// <param name="page">The page to get.
00942 /// </param>
00943 /// </returns>
00944 public void SetPage(int index, Page page)
00945 {
00946     Pages[index] = page;
00947 }
00948
00949 /// <summary>
00950 /// Get the page at the specified index.
00951 /// </summary>
00952 /// <param name="index">The index of the page to get.
00953 /// </param>
00954 /// <param name="page">The page to get.
00955 /// </param>
00956 /// </returns>
00957 public Page GetPage(int index)
00958 {
00959     return Pages[index];
00960 }
00961
00962 /// <summary>
00963 /// Get the page at the specified index.
00964 /// </summary>
00965 /// <param name="index">The index of the page to get.
00966 /// </param>
00967 /// <param name="page">The page to get.
00968 /// </param>
00969 /// </returns>
00970 public void SetPage(int index, Page page)
00971 {
00972     Pages[index] = page;
00973 }
00974
00975 /// <summary>
00976 /// Get the page at the specified index.
00977 /// </summary>
00978 /// <param name="index">The index of the page to get.
00979 /// </param>
00980 /// <param name="page">The page to get.
00981 /// </param>
00982 /// </returns>
00983 public Page GetPage(int index)
00984 {
00985     return Pages[index];
00986 }
00987
00988 /// <summary>
00989 /// Get the page at the specified index.
00990 /// </summary>
00991 /// <param name="index">The index of the page to get.
00992 /// </param>
00993 /// <param name="page">The page to get.
00994 /// </param>
00995 /// </returns>
00996 public void SetPage(int index, Page page)
00997 {
00998     Pages[index] = page;
00999 }
01000
01001 /// <summary>
01002 /// Get the page at the specified index.
01003 /// </summary>
01004 /// <param name="index">The index of the page to get.
01005 /// </param>
01006 /// <param name="page">The page to get.
01007 /// </param>
01008 /// </returns>
01009 public Page GetPage(int index)
01010 {
01011     return Pages[index];
01012 }
01013
01014 /// <summary>
01015 /// Get the page at the specified index.
01016 /// </summary>
01017 /// <param name="index">The index of the page to get.
01018 /// </param>
01019 /// <param name="page">The page to get.
01020 /// </param>
01021 /// </returns>
01022 public void SetPage(int index, Page page)
01023 {
01024     Pages[index] = page;
01025 }
01026
01027 /// <summary>
01028 /// Get the page at the specified index.
01029 /// </summary>
01030 /// <param name="index">The index of the page to get.
01031 /// </param>
01032 /// <param name="page">The page to get.
01033 /// </param>
01034 /// </returns>
01035 public Page GetPage(int index)
01036 {
01037     return Pages[index];
01038 }
01039
01040 /// <summary>
01041 /// Get the page at the specified index.
01042 /// </summary>
01043 /// <param name="index">The index of the page to get.
01044 /// </param>
01045 /// <param name="page">The page to get.
01046 /// </param>
01047 /// </returns>
01048 public void SetPage(int index, Page page)
01049 {
01050     Pages[index] = page;
01051 }
01052
01053 /// <summary>
01054 /// Get the page at the specified index.
01055 /// </summary>
01056 /// <param name="index">The index of the page to get.
01057 /// </param>
01058 /// <param name="page">The page to get.
01059 /// </param>
01060 /// </returns>
01061 public Page GetPage(int index)
01062 {
01063     return Pages[index];
01064 }
01065
01066 /// <summary>
01067 /// Get the page at the specified index.
01068 /// </summary>
01069 /// <param name="index">The index of the page to get.
01070 /// </param>
01071 /// <param name="page">The page to get.
01072 /// </param>
01073 /// </returns>
01074 public void SetPage(int index, Page page)
01075 {
01076     Pages[index] = page;
01077 }
01078
01079 /// <summary>
01080 /// Get the page at the specified index.
01081 /// </summary>
01082 /// <param name="index">The index of the page to get.
01083 /// </param>
01084 /// <param name="page">The page to get.
01085 /// </param>
01086 /// </returns>
01087 public Page GetPage(int index)
01088 {
01089     return Pages[index];
01090 }
01091
01092 /// <summary>
01093 /// Get the page at the specified index.
01094 /// </summary>
01095 /// <param name="index">The index of the page to get.
01096 /// </param>
01097 /// <param name="page">The page to get.
01098 /// </param>
01099 /// </returns>
01100 public void SetPage(int index, Page page)
01101 {
01102     Pages[index] = page;
01103 }
01104
01105 /// <summary>
01106 /// Get the page at the specified index.
01107 /// </summary>
01108 /// <param name="index">The index of the page to get.
01109 /// </param>
01110 /// <param name="page">The page to get.
01111 /// </param>
01112 /// </returns>
01113 public Page GetPage(int index)
01114 {
01115     return Pages[index];
01116 }
01117
01118 /// <summary>
01119 /// Get the page at the specified index.
01120 /// </summary>
01121 /// <param name="index">The index of the page to get.
01122 /// </param>
01123 /// <param name="page">The page to get.
01124 /// </param>
01125 /// </returns>
01126 public void SetPage(int index, Page page)
01127 {
01128     Pages[index] = page;
01129 }
01130
01131 /// <summary>
01132 /// Get the page at the specified index.
01133 /// </summary>
01134 /// <param name="index">The index of the page to get.
01135 /// </param>
01136 /// <param name="page">The page to get.
01137 /// </param>
01138 /// </returns>
01139 public Page GetPage(int index)
01140 {
01141     return Pages[index];
01142 }
01143
01144 /// <summary>
01145 /// Get the page at the specified index.
01146 /// </summary>
01147 /// <param name="index">The index of the page to get.
01148 /// </param>
01149 /// <param name="page">The page to get.
01150 /// </param>
01151 /// </returns>
01152 public void SetPage(int index, Page page)
01153 {
01154     Pages[index] = page;
01155 }
01156
01157 /// <summary>
01158 /// Get the page at the specified index.
01159 /// </summary>
01160 /// <param name="index">The index of the page to get.
01161 /// </param>
01162 /// <param name="page">The page to get.
01163 /// </param>
01164 /// </returns>
01165 public Page GetPage(int index)
01166 {
01167     return Pages[index];
01168 }
01169
01170 /// <summary>
01171 /// Get the page at the specified index.
01172 /// </summary>
01173 /// <param name="index">The index of the page to get.
01174 /// </param>
01175 /// <param name="page">The page to get.
01176 /// </param>
01177 /// </returns>
01178 public void SetPage(int index, Page page)
01179 {
01180     Pages[index] = page;
01181 }
01182
01183 /// <summary>
01184 /// Get the page at the specified index.
01185 /// </summary>
01186 /// <param name="index">The index of the page to get.
01187 /// </param>
01188 /// <param name="page">The page to get.
01189 /// </param>
01190 /// </returns>
01191 public Page GetPage(int index)
01192 {
01193     return Pages[index];
01194 }
01195
01196 /// <summary>
01197 /// Get the page at the specified index.
01198 /// </summary>
01199 /// <param name="index">The index of the page to get.
01200 /// </param>
01201 /// <param name="page">The page to get.
01202 /// </param>
01203 /// </returns>
01204 public void SetPage(int index, Page page)
01205 {
01206     Pages[index] = page;
01207 }
01208
01209 /// <summary>
01210 /// Get the page at the specified index.
01211 /// </summary>
01212 /// <param name="index">The index of the page to get.
01213 /// </param>
01214 /// <param name="page">The page to get.
01215 /// </param>
01216 /// </returns>
01217 public Page GetPage(int index)
01218 {
01219     return Pages[index];
01220 }
01221
01222 /// <summary>
01223 /// Get the page at the specified index.
01224 /// </summary>
01225 /// <param name="index">The index of the page to get.
01226 /// </param>
01227 /// <param name="page">The page to get.
01228 /// </param>
01229 /// </returns>
01230 public void SetPage(int index, Page page)
01231 {
01232     Pages[index] = page;
01233 }
01234
01235 /// <summary>
01236 /// Get the page at the specified index.
01237 /// </summary>
01238 /// <param name="index">The index of the page to get.
01239 /// </param>
01240 /// <param name="page">The page to get.
01241 /// </param>
01242 /// </returns>
01243 public Page GetPage(int index)
01244 {
01245     return Pages[index];
01246 }
01247
01248 /// <summary>
01249 /// Get the page at the specified index.
01250 /// </summary>
01251 /// <param name="index">The index of the page to get.
01252 /// </param>
01253 /// <param name="page">The page to get.
01254 /// </param>
01255 /// </returns>
01256 public void SetPage(int index, Page page)
01257 {
01258     Pages[index] = page;
01259 }
01260
01261 /// <summary>
01262 /// Get the page at the specified index.
01263 /// </summary>
01264 /// <param name="index">The index of the page to get.
01265 /// </param>
01266 /// <param name="page">The page to get.
01267 /// </param>
01268 /// </returns>
01269 public Page GetPage(int index)
01270 {
01271     return Pages[index];
01272 }
01273
01274 /// <summary>
01275 /// Get the page at the specified index.
01276 /// </summary>
01277 /// <param name="index">The index of the page to get.
01278 /// </param>
01279 /// <param name="page">The page to get.
01280 /// </param>
01281 /// </returns>
01282 public void SetPage(int index, Page page)
01283 {
01284     Pages[index] = page;
01285 }
01286
01287 /// <summary>
01288 /// Get the page at the specified index.
01289 /// </summary>
01290 /// <param name="index">The index of the page to get.
01291 /// </param>
01292 /// <param name="page">The page to get.
01293 /// </param>
01294 /// </returns>
01295 public Page GetPage(int index)
01296 {
01297     return Pages[index];
01298 }
01299
01300 /// <summary>
01301 /// Get the page at the specified index.
01302 /// </summary>
01303 /// <param name="index">The index of the page to get.
01304 /// </param>
01305 /// <param name="page">The page to get.
01306 /// </param>
01307 /// </returns>
01308 public void SetPage(int index, Page page)
01309 {
01310     Pages[index] = page;
01311 }
01312
01313 /// <summary>
01314 /// Get the page at the specified index.
01315 /// </summary>
01316 /// <param name="index">The index of the page to get.
01317 /// </param>
01318 /// <param name="page">The page to get.
01319 /// </param>
01320 /// </returns>
01321 public Page GetPage(int index)
01322 {
01323     return Pages[index];
01324 }
01325
01326 /// <summary>
01327 /// Get the page at the specified index.
01328 /// </summary>
01329 /// <param name="index">The index of the page to get.
01330 /// </param>
01331 /// <param name="page">The page to get.
01332 /// </param>
01333 /// </returns>
01334 public void SetPage(int index, Page page)
01335 {
01336     Pages[index] = page;
01337 }
01338
01339 /// <summary>
01340 /// Get the page at the specified index.
01341 /// </summary>
01342 /// <param name="index">The index of the page to get.
01343 /// </param>
01344 /// <param name="page">The page to get.
01345 /// </param>
01346 /// </returns>
01347 public Page GetPage(int index)
01348 {
01349     return Pages[index];
01350 }
01351
01352 /// <summary>
01353 /// Get the page at the specified index.
01354 /// </summary>
01355 /// <param name="index">The index of the page to get.
01356 /// </param>
01357 /// <param name="page">The page to get.
01358 /// </param>
01359 /// </returns>
01360 public void SetPage(int index, Page page)
01361 {
01362     Pages[index] = page;
01363 }
01364
01365 /// <summary>
01366 /// Get the page at the specified index.
01367 /// </summary>
01368 /// <param name="index">The index of the page to get.
01369 /// </param>
01370 /// <param name="page">The page to get.
01371 /// </param>
01372 /// </returns>
01373 public Page GetPage(int index)
01374 {
01375     return Pages[index];
01376 }
01377
01378 /// <summary>
01379 /// Get the page at the specified index.
01380 /// </summary>
01381 /// <param name="index">The index of the page to get.
01382 /// </param>
01383 /// <param name="page">The page to get.
01384 /// </param>
01385 /// </returns>
01386 public void SetPage(int index, Page page)
01387 {
01388     Pages[index] = page;
01389 }
01390
01391 /// <summary>
01392 /// Get the page at the specified index.
01393 /// </summary>
01394 /// <param name="index">The index of the page to get.
01395 /// </param>
01396 /// <param name="page">The page to get.
01397 /// </param>
01398 /// </returns>
01399 public Page GetPage(int index)
01400 {
01401     return Pages[index];
01402 }
01403
01404 /// <summary>
01405 /// Get the page at the specified index.
01406 /// </summary>
01407 /// <param name="index">The index of the page to get.
01408 /// </param>
01409 /// <param name="page">The page to get.
01410 /// </param>
01411 /// </returns>
01412 public void SetPage(int index, Page page)
01413 {
01414     Pages[index] = page;
01415 }
01416
01417 /// <summary>
01418 /// Get the page at the specified index.
01419 /// </summary>
01420 /// <param name="index">The index of the page to get.
01421 /// </param>
01422 /// <param name="page">The page to get.
01423 /// </param>
01424 /// </returns>
01425 public Page GetPage(int index)
01426 {
01427     return Pages[index];
01428 }
01429
01430 /// <summary>
01431 /// Get the page at the specified index.
01432 /// </summary>
01433 /// <param name="index">The index of the page to get.
01434 /// </param>
01435 /// <param name="page">The page to get.
01436 /// </param>
01437 /// </returns>
01438 public void SetPage(int index, Page page)
01439 {
01440     Pages[index] = page;
01441 }
01442
01443 /// <summary>
01444 /// Get the page at the specified index.
01445 /// </summary>
01446 /// <param name="index">The index of the page to get.
01447 /// </param>
01448 /// <param name="page">The page to get.
01449 /// </param>
01450 /// </returns>
01451 public Page GetPage(int index)
01452 {
01453     return Pages[index];
01454 }
01455
01456 /// <summary>
01457 /// Get the page at the specified index.
01458 /// </summary>
01459 /// <param name="index">The index of the page to get.
01460 /// </param>
01461 /// <param name="page">The page to get.
01462 /// </param>
01463 /// </returns>
01464 public void SetPage(int index, Page page)
01465 {
01466     Pages[index] = page;
01467 }
01468
01469 /// <summary>
01470 /// Get the page at the specified index.
01471 /// </summary>
01472 /// <param name="index">The index of the page to get.
01473 /// </param>
01474 /// <param name="page">The page to get.
01475 /// </param>
01476 /// </returns>
01477 public Page GetPage(int index)
01478 {
01479     return Pages[index];
01480 }
01481
01482 /// <summary>
01483 /// Get the page at the specified index.
01484 /// </summary>
01485 /// <param name="index">The index of the page to get.
01486 /// </param>
01
```

```

00689         default:
00690             throw new MuPDFException("Unknown error", result);
00691     }
00692 
00693     RoundedRectangle roundedRegion = region.Round(fzoom);
00694     RoundedSize roundedSize = new RoundedSize(roundedRegion.Width, roundedRegion.Height);
00695 
00696     if (pixelFormat == PixelFormats.RGBA || pixelFormat == PixelFormats.BGRA)
00697     {
00698         Utils.UnpremultiplyAlpha(destination, roundedSize);
00699     }
00700 
00701     if (this.ClipToPageBounds &&
00702     !Pages[pageNumber].Bounds.Contains(DisplayLists[pageNumber].Bounds.Intersect(region)))
00703     {
00704         Utils.ClipImage(destination, roundedSize, region, Pages[pageNumber].Bounds,
00705         pixelFormat);
00706     }
00707 
00708     /// <summary>
00709     /// Render a page to the specified destination.
00710     /// </summary>
00711     /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00712     /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00713     /// size in pixel of the image.</param>
00714     /// <param name="pixelFormat">The format of the pixel data.</param>
00715     /// <param name="destination">The address of the buffer where the pixel data will be written.
00716     There must be enough space available to write the values for all the pixels, otherwise this will fail
00717     catastrophically!</param>
00718     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00719     signatures) are included in the display list that is generated. Otherwise, only the page contents are
00720     included.</param>
00721     public void Render(int pageNumber, double zoom, PixelFormats pixelFormat, IntPtr destination,
00722     bool includeAnnotations = true)
00723     {
00724         if (this.EncryptionState == EncryptionState.Encrypted)
00725         {
00726             throw new DocumentLockedException("A password is necessary to render the document!");
00727         }
00728 
00729         Rectangle region = this.Pages[pageNumber].Bounds;
00730         Render(pageNumber, region, zoom, pixelFormat, destination, includeAnnotations);
00731     }
00732 
00733     /// <summary>
00734     /// Render (part of) a page to a <see cref="Span{T}">Span</see>&lt;;<see cref="byte"/>&gt;.
00735     /// </summary>
00736     /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00737     /// <param name="region">The region of the page to render in page units.</param>
00738     /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00739     /// size in pixel of the image.</param>
00740     /// <param name="pixelFormat">The format of the pixel data.</param>
00741     /// <param name="disposable">An <see cref="IDisposable"/> that can be used to free the memory
00742     where the image is stored. You should keep track of this and dispose it when you have finished
00743     working with the image.</param>
00744     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00745     signatures) are included in the display list that is generated. Otherwise, only the page contents are
00746     included.</param>
00747     public Span<byte> Render(int pageNumber, Rectangle region, double zoom, PixelFormats
00748     pixelFormat, out IDisposable disposable, bool includeAnnotations = true)
00749     {
00750         if (this.EncryptionState == EncryptionState.Encrypted)
00751         {
00752             throw new DocumentLockedException("A password is necessary to render the document!");
00753         }
00754 
00755         int dataSize = GetRenderedSize(region, zoom, pixelFormat);
00756 
00757         IntPtr destination = Marshal.AllocHGlobal(dataSize);
00758         disposable = new DisposableIntPtr(destination, dataSize);
00759 
00760         this.Render(pageNumber, region, zoom, pixelFormat, destination, includeAnnotations);
00761 
00762         unsafe
00763         {
00764             return new Span<byte>((void*)destination, dataSize);
00765         }
00766     }
00767 
00768     /// <summary>
00769     /// Render a page to a <see cref="Span{T}">Span</see>&lt;;<see cref="byte"/>&gt;.
00770     /// </summary>
00771     /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00772     /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00773     /// size in pixel of the image.</param>
00774     /// <param name="pixelFormat">The format of the pixel data.</param>

```

```

00761     /// <param name="disposable">An <see cref="IDisposable"/> that can be used to free the memory
00762     where the image is stored. You should keep track of this and dispose it when you have finished
00763     working with the image.</param>
00764     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00765     signatures) are included in the display list that is generated. Otherwise, only the page contents are
00766     included.</param>
00767     public Span<byte> Render(int pageNumber, double zoom, PixelFormats pixelFormat, out
00768     IDisposable disposable, bool includeAnnotations = true)
00769     {
00770         if (this.EncryptionState == EncryptionState.Encrypted)
00771         {
00772             throw new DocumentLockedException("A password is necessary to render the document!");
00773         }
00774
00775         Rectangle region = this.Pages[pageNumber].Bounds;
00776         return Render(pageNumber, region, zoom, pixelFormat, out disposable, includeAnnotations);
00777     }
00778
00779     /// <summary>
00780     /// Create a new <see cref="MuPDFMultiThreadedPageRenderer"/> that renders the specified page
00781     with the specified number of threads.
00782     /// </summary>
00783     /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00784     /// <param name="threadCount">The number of threads to use. This must be factorisable using
00785     only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than <paramref
00786     name="threadCount"/> that satisfies this condition is used.</param>
00787     /// <returns>A <see cref="MuPDFMultiThreadedPageRenderer"/> that can be used to render the
00788     specified page with the specified number of threads.</returns>
00789     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00790     signatures) are included in the display list that is generated. Otherwise, only the page contents are
00791     included.</param>
00792     public MuPDFMultiThreadedPageRenderer GetMultiThreadedRenderer(int pageNumber, int
00793     threadCount, bool includeAnnotations = true)
00794     {
00795         if (this.EncryptionState == EncryptionState.Encrypted)
00796         {
00797             throw new DocumentLockedException("A password is necessary to render the document!");
00798         }
00799
00800         if (DisplayLists[pageNumber] == null)
00801         {
00802             DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
00803             this.Pages[pageNumber], includeAnnotations);
00804         }
00805
00806         return new MuPDFMultiThreadedPageRenderer(OwnerContext, DisplayLists[pageNumber],
00807         threadCount, Pages[pageNumber].Bounds, this.ClipToPageBounds, this.ImageXRes, this.ImageYRes);
00808     }
00809
00810     /// <summary>
00811     /// Determine how many bytes will be necessary to render the specified page at the specified
00812     zoom level, using the the specified pixel format.
00813     /// </summary>
00814     /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00815     /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00816     size in pixel of the image.</param>
00817     /// <param name="pixelFormat">The format of the pixels data.</param>
00818     /// <returns>An integer representing the number of bytes that will be necessary to store the
00819     pixel data of the rendered image.</returns>
00820     public int GetRenderedSize(int pageNumber, double zoom, PixelFormats pixelFormat)
00821     {
00822         if (this.EncryptionState == EncryptionState.Encrypted)
00823         {
00824             throw new DocumentLockedException("A password is necessary to render the document!");
00825         }
00826
00827         return GetRenderedSize(Pages[pageNumber].Bounds, zoom, pixelFormat);
00828     }
00829
00830     /// <summary>
00831     /// Determine how many bytes will be necessary to render the specified region in page units at
00832     the specified zoom level, using the the specified pixel format.
00833     /// </summary>
00834     /// <param name="region">The region that will be rendered.</param>
00835     /// <param name="zoom">The scale at which the region will be rendered. This will determine the
00836     size in pixel of the image.</param>
00837     /// <param name="pixelFormat">The format of the pixels data.</param>
00838     /// <returns>An integer representing the number of bytes that will be necessary to store the
00839     pixel data of the rendered image.</returns>
00840     public static int GetRenderedSize(Rectangle region, double zoom, PixelFormats pixelFormat)
00841     {
00842         float x0 = region.X0 * (float)zoom;
00843         float y0 = region.Y0 * (float)zoom;
00844         float x1 = region.X1 * (float)zoom;
00845         float y1 = region.Y1 * (float)zoom;
00846     }

```

```

00828     Rectangle scaledRect = new Rectangle(x0, y0, x1, y1);
00829     RoundedRectangle bounds = scaledRect.Round();
00830
00831     int width = bounds.Width;
00832     int height = bounds.Height;
00833
00834     switch (pixelFormat)
00835     {
00836         case PixelFormats.RGB:
00837         case PixelFormats.BGR:
00838             return width * height * 3;
00839         case PixelFormats.RGBA:
00840         case PixelFormats.BGRA:
00841             return width * height * 4;
00842     }
00843
00844     return -1;
00845 }
00846
00847 /// <summary>
00848 /// Save (part of) a page to an image file in the specified format.
00849 /// </summary>
00850 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00851 /// <param name="region">The region of the page to render in page units.</param>
00852 /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00853 /// size in pixel of the image.</param>
00854 /// <param name="pixelFormat">The format of the pixel data.</param>
00855 /// <param name="fileName">The path to the output file.</param>
00856 /// <param name="fileType">The output format of the file.</param>
00857 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00858 signatures) are included in the display list that is generated. Otherwise, only the page contents are
00859 included.</param>
00860     public void SaveImage(int pageNumber, Rectangle region, double zoom, PixelFormats pixelFormat,
00861 string fileName, RasterOutputFileTypes fileType, bool includeAnnotations = true)
00862     {
00863         if (this.EncryptionState == EncryptionState.Encrypted)
00864         {
00865             throw new DocumentLockedException("A password is necessary to render the document!");
00866         }
00867
00868         if (pixelFormat == PixelFormats.RGBA && fileType == RasterOutputFileTypes.PNM)
00869         {
00870             throw new ArgumentException("Cannot save an image with alpha channel in PNM format!",
00871 nameof(fileType));
00872         }
00873
00874         if (DisplayLists[pageNumber] == null)
00875         {
00876             DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
00877 this.Pages[pageNumber], includeAnnotations);
00878         }
00879
00880         if (zoom < 0.000001 || zoom * region.Width <= 0.001 || zoom * region.Height <= 0.001)
00881         {
00882             throw new ArgumentOutOfRangeException(nameof(zoom), zoom, "The zoom factor is too
00883 small!");
00884         }
00885
00886         if (this.ImageXRes != 72 || this.ImageYRes != 72)
00887         {
00888             zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
00889             region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 / this.ImageYRes,
00890             region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
00891         }
00892
00893         float fzoom = (float)zoom;
00894
00895         ExitCodes result;
00896
00897         using (UTF8EncodedString encodedFileName = new UTF8EncodedString(fileName))
00898         {
00899             result = (ExitCodes)NativeMethods.SaveImage(OwnerContext.NativeContext,
01000             DisplayLists[pageNumber].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, fzoom,
01001             (int)pixelFormat, encodedFileName.Address, (int)fileType);
01002         }
01003
01004         switch (result)
01005         {
01006             case ExitCodes.EXIT_SUCCESS:
01007                 break;
01008             case ExitCodes.ERR_CANNOT_RENDER:
01009                 throw new MuPDFException("Cannot render page", result);
01010             case ExitCodes.ERR_CANNOT_SAVE:
01011                 throw new MuPDFException("Cannot save to the output file", result);
01012             default:
01013                 throw new MuPDFException("Unknown error", result);
01014         }
01015     }

```

```

00905         }
00906
00907     /// <summary>
00908     /// Save a page to an image file in the specified format.
00909     /// </summary>
00910     /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00911     /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00912     /// size in pixel of the image.</param>
00913     /// <param name="pixelFormat">The format of the pixel data.</param>
00914     /// <param name="fileName">The path to the output file.</param>
00915     /// <param name="fileType">The output format of the file.</param>
00916     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00917     /// signatures) are included in the display list that is generated. Otherwise, only the page contents are
00918     /// included.</param>
00919     public void SaveImage(int pageNumber, double zoom, PixelFormats pixelFormat, string fileName,
00920     RasterOutputFileTypes fileType, bool includeAnnotations = true)
00921     {
00922         if (this.EncryptionState == EncryptionState.Encrypted)
00923         {
00924             throw new DocumentLockedException("A password is necessary to render the document!");
00925         }
00926
00927         Rectangle region = this.Pages[pageNumber].Bounds;
00928         SaveImage(pageNumber, region, zoom, pixelFormat, fileName, fileType, includeAnnotations);
00929     }
00930
00931     /// <summary>
00932     /// Write (part of) a page to an image stream in the specified format.
00933     /// </summary>
00934     /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
00935     /// <param name="region">The region of the page to render in page units.</param>
00936     /// <param name="zoom">The scale at which the page will be rendered. This will determine the
00937     /// size in pixel of the image.</param>
00938     /// <param name="pixelFormat">The format of the pixel data.</param>
00939     /// <param name="outputStream">The stream to which the image data will be written.</param>
00940     /// <param name="fileType">The output format of the image.</param>
00941     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
00942     /// signatures) are included in the display list that is generated. Otherwise, only the page contents are
00943     /// included.</param>
00944     public void WriteImage(int pageNumber, Rectangle region, double zoom, PixelFormats
00945     pixelFormat, Stream outputStream, RasterOutputFileTypes fileType, bool includeAnnotations = true)
00946     {
00947         if (this.EncryptionState == EncryptionState.Encrypted)
00948         {
00949             throw new DocumentLockedException("A password is necessary to render the document!");
00950         }
00951
00952         if (pixelFormat == PixelFormats.RGBA && fileType == RasterOutputFileTypes.PNM)
00953         {
00954             throw new ArgumentException("Cannot save an image with alpha channel in PNM format!",
00955             nameof(fileType));
00956         }
00957
00958         if (DisplayLists[pageNumber] == null)
00959         {
00960             DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
00961             this.Pages[pageNumber], includeAnnotations);
00962         }
00963
00964         if (zoom < 0.000001 || zoom * region.Width <= 0.001 || zoom * region.Height <= 0.001)
00965         {
00966             throw new ArgumentOutOfRangeException(nameof(zoom), zoom, "The zoom factor is too
00967             small!");
00968         }
00969
00970         if (this.ImageXRes != 72 || this.ImageYRes != 72)
00971         {
00972             zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
00973             region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 / this.ImageYRes,
00974             region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
00975         }
00976
00977         float fzoom = (float)zoom;
00978
00979         IntPtr outputBuffer = IntPtr.Zero;
00980         IntPtr outputData = IntPtr.Zero;
00981         ulong outputDataLength = 0;
00982
00983         ExitCodes result = (ExitCodes)NativeMethods.WriteImage(OwnerContext.NativeContext,
00984             DisplayLists[pageNumber].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, fzoom,
00985             (int)pixelFormat, (int)fileType, ref outputBuffer, ref outputData, ref outputDataLength);
00986
00987         switch (result)
00988         {
00989             case ExitCodes.EXIT_SUCCESS:
00990                 break;
00991             case ExitCodes.ERR_CANNOT_RENDER:
00992                 break;
00993         }
00994     }
00995 
```

```

00978             throw new MuPDFException("Cannot render page", result);
00979         case ExitCodes.ERR_CANNOT_CREATE_CONTEXT:
00980             throw new MuPDFException("Cannot create the output buffer", result);
00981         default:
00982             throw new MuPDFException("Unknown error", result);
00983     }
00984 
00985     byte[] buffer = new byte[1024];
00986 
00987     while (outputDataLength > 0)
00988     {
00989         int bytesToCopy = (int)Math.Min(buffer.Length, (long)outputDataLength);
00990         Marshal.Copy(outputData, buffer, 0, bytesToCopy);
00991         outputData = IntPtr.Add(outputData, bytesToCopy);
00992         outputStream.Write(buffer, 0, bytesToCopy);
00993         outputDataLength -= (ulong)bytesToCopy;
00994     }
00995 
00996     NativeMethods.DisposeBuffer(OwnerContext.NativeContext, outputBuffer);
00997 }
00998 
00999 /// <summary>
01000 /// Write a page to an image stream in the specified format.
01001 /// </summary>
01002 /// <param name="pageNumber">The number of the page to render (starting at 0).</param>
01003 /// <param name="zoom">The scale at which the page will be rendered. This will determine the
01004 /// size in pixel of the image.</param>
01005 /// <param name="pixelFormat">The format of the pixel data.</param>
01006 /// <param name="outputStream">The stream to which the image data will be written.</param>
01007 /// <param name="fileType">The output format of the image.</param>
01008 /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01009 signatures) are included in the display list that is generated. Otherwise, only the page contents are
01010 included.</param>
01011 public void WriteImage(int pageNumber, double zoom, PixelFormats pixelFormat, Stream
01012 outputStream, RasterOutputFileTypes fileType, bool includeAnnotations = true)
01013 {
01014     if (this.EncryptionState == EncryptionState.Encrypted)
01015     {
01016         throw new DocumentLockedException("A password is necessary to render the document!");
01017     }
01018 
01019     Rectangle region = this.Pages[pageNumber].Bounds;
01020     WriteImage(pageNumber, region, zoom, pixelFormat, outputStream, fileType,
01021     includeAnnotations);
01022 
01023     /// <summary>
01024     /// Create a new document containing the specified (parts of) pages from other documents.
01025     /// </summary>
01026     /// <param name="context">The context that was used to open the documents.</param>
01027     /// <param name="fileName">The output file name.</param>
01028     /// <param name="fileType">The output file format.</param>
01029     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01030 signatures) are included in the display list that is generated. Otherwise, only the page contents are
01031 included.</param>
01032     /// <param name="pages">The pages to include in the document. The "page" element specifies the
01033     /// page, the "region" element the area of the page that should be included in the document, and the
01034     /// "zoom" element how much the region should be scaled.</param>
01035     public static void CreateDocument(MuPDFContext context, string fileName,
01036     DocumentOutputFileTypes fileType, bool includeAnnotations = true, params (MuPDFPage page, Rectangle
01037     region, float zoom)[] pages)
01038     {
01039         if (fileType == DocumentOutputFileTypes.SVG && pages.Length > 1)
01040         {
01041             //Actually, you can, but the library creates multiple files appending numbers after
01042             //each name (e.g. pagel.svg, page2.svg, ...), which is ugly and may have unintended consequences.
01043             //If you really want to do this, you can call this method multiple times.
01044             throw new ArgumentException("You cannot create an SVG document with more than one
01045             page!", nameof(pages));
01046         }
01047 
01048         string originalFileName = fileName;
01049 
01050         if (fileType == DocumentOutputFileTypes.SVG)
01051         {
01052             //For SVG documents, the library annoyingly alters the output file name, appending a
01053             //"1" just before the extension (e.g. document.svg -> document1.svg). Since users may not be expecting
01054             //this, it is best to render to a temporary file and then move it to the specified location.
01055             fileName = Path.GetTempFileName();
01056         }
01057 
01058         IntPtr documentWriter = IntPtr.Zero;
01059 
01060         ExitCodes result;
01061 
01062         // Encode the file name in UTF-8 in unmanaged memory.
01063         using (UTF8EncodedString encodedFileName = new UTF8EncodedString(fileName))

```

```

01050         {
01051             //Initialise document writer.
01052             result = (ExitCodes)NativeMethods.CreateDocumentWriter(context.NativeContext,
01053             encodedFileName.Address, (int)fileType, ref documentWriter);
01054         }
01055         switch (result)
01056     {
01057         case ExitCodes.EXIT_SUCCESS:
01058             break;
01059         case ExitCodes.ERR_CANNOT_CREATE_WRITER:
01060             throw new MuPDFException("Cannot create the document writer", result);
01061         default:
01062             throw new MuPDFException("Unknown error", result);
01063     }
01064
01065     //Write pages.
01066     for (int i = 0; i < pages.Length; i++)
01067     {
01068         MuPDFDocument doc = pages[i].page.OwnerDocument;
01069         int pageNum = pages[i].page.PageNumber;
01070
01071         if (doc.DisplayLists[pageNum] == null)
01072         {
01073             doc.DisplayLists[pageNum] = new MuPDFDisplayList(doc.OwnerContext,
01074             doc.Pages[pageNum], includeAnnotations);
01075         }
01076
01077         Rectangle region = pages[i].region;
01078         double zoom = pages[i].zoom;
01079
01080         if (pages[i].page.OwnerDocument.ImageXRes != 72 ||
01081             pages[i].page.OwnerDocument.ImageYRes != 72)
01082         {
01083             zoom *= Math.Sqrt(pages[i].page.OwnerDocument.ImageXRes *
01084                 pages[i].page.OwnerDocument.ImageYRes) / 72;
01085             region = new Rectangle(region.X0 * 72 / pages[i].page.OwnerDocument.ImageXRes,
01086             region.Y0 * 72 / pages[i].page.OwnerDocument.ImageYRes, region.X1 * 72 /
01087             pages[i].page.OwnerDocument.ImageXRes, region.Y1 * 72 / pages[i].page.OwnerDocument.ImageYRes);
01088         }
01089
01090         result = (ExitCodes)NativeMethods.WriteSubDisplayListAsPage(context.NativeContext,
01091             doc.DisplayLists[pageNum].NativeDisplayList, region.X0, region.Y0, region.X1, region.Y1, (float)zoom,
01092             documentWriter);
01093
01094         switch (result)
01095     {
01096         case ExitCodes.EXIT_SUCCESS:
01097             break;
01098         case ExitCodes.ERR_CANNOT_RENDER:
01099             throw new MuPDFException("Cannot render page " + i.ToString(), result);
01100         default:
01101             throw new MuPDFException("Unknown error", result);
01102     }
01103
01104     //Close and dispose the document writer.
01105     result = (ExitCodes)NativeMethods.FinalizeDocumentWriter(context.NativeContext,
01106             documentWriter);
01107
01108     switch (result)
01109     {
01110         case ExitCodes.EXIT_SUCCESS:
01111             break;
01112         case ExitCodes.ERR_CANNOT_CLOSE_DOCUMENT:
01113             throw new MuPDFException("Cannot finalise the document", result);
01114         default:
01115             throw new MuPDFException("Unknown error", result);
01116     }
01117
01118     if (fileType == DocumentOutputFileTypes.SVG)
01119     {
01120         //Move the temporary file to the location specified by the user.
01121         //The library has altered the temporary file name by appending a "1" before the
01122         //extension.
01123         string tempFileName = Path.Combine(Path.GetDirectoryName(fileName),
01124             Path.GetFileNameWithoutExtension(fileName) + "1" + Path.GetExtension(fileName));
01125
01126         //Overwrite existing file.
01127         if (File.Exists(originalFileName))
01128         {
01129             File.Delete(originalFileName);
01130         }
01131
01132         File.Move(tempFileName, originalFileName);
01133     }
01134 }

```

```

01126
01127    /// <summary>
01128    /// Create a new document containing the specified pages from other documents.
01129    /// </summary>
01130    /// <param name="context">The context that was used to open the documents.</param>
01131    /// <param name="fileName">The output file name.</param>
01132    /// <param name="fileType">The output file format.</param>
01133    /// <param name="pages">The pages to include in the document.</param>
01134    /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01135    signatures) are included in the display list that is generated. Otherwise, only the page contents are
01136    included.</param>
01137    public static void CreateDocument(MuPDFContext context, string fileName,
01138        DocumentOutputFileTypes fileType, bool includeAnnotations = true, params MuPDFPage[] pages)
01139    {
01140        (MuPDFPage, Rectangle, float)[] boundedPages = new (MuPDFPage, Rectangle,
01141        float)[pages.Length];
01142
01143        for (int i = 0; i < pages.Length; i++)
01144        {
01145            boundedPages[i] = (pages[i], pages[i].Bounds, 1);
01146        }
01147
01148        CreateDocument(context, fileName, fileType, includeAnnotations, boundedPages);
01149    }
01150
01151    /// <summary>
01152    /// Creates a new <see cref="MuPDFStructuredTextPage"/> from the specified page. This contains
01153    information about the text layout that can be used for highlighting and searching. The reading order
01154    is taken from the order the text is drawn in the source file, so may not be accurate.
01155    /// </summary>
01156    /// <param name="pageNumber">The number of the page (starting at 0)</param>
01157    /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01158    signatures) are included. Otherwise, only the page contents are included.</param>
01159    /// <returns>A <see cref="MuPDFStructuredTextPage"/> containing a structured text
01160    representation of the page.</returns>
01161    public MuPDFStructuredTextPage GetStructuredTextPage(int pageNumber, bool includeAnnotations =
01162        true)
01163    {
01164        if (this.EncryptionState == EncryptionState.Encrypted)
01165        {
01166            throw new DocumentLockedException("A password is necessary to render the document!");
01167        }
01168
01169        if (DisplayLists[pageNumber] == null)
01170        {
01171            DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
01172                this.Pages[pageNumber], includeAnnotations);
01173
01174            return new MuPDFStructuredTextPage(this.OwnerContext, this.DisplayLists[pageNumber], null,
01175                1, new Rectangle());
01176        }
01177
01178        /// <summary>
01179        /// Creates a new <see cref="MuPDFStructuredTextPage"/> from the specified page, using optical
01180        character recognition (OCR) to determine what text is written on the image. This contains information
01181        about the text layout that can be used for highlighting and searching.
01182        /// </summary>
01183        /// <param name="pageNumber">The number of the page (starting at 0)</param>
01184        /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
01185        this is null, no OCR is performed.</param>
01186        /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01187        signatures) are included. Otherwise, only the page contents are included.</param>
01188        /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
01189        operation. Providing a value other than the default is not supported on Windows x86 and will throw a
01190        runtime exception.</param>
01191        /// <param name="progress">An <see cref="IProgress<OCRProgressInfo>"/> used to report
01192        progress. Providing a value other than null is not supported on Windows x86 and will throw a runtime
01193        exception.</param>
01194        /// <returns>A <see cref="MuPDFStructuredTextPage"/> containing a structured text
01195        representation of the page.</returns>
01196        public MuPDFStructuredTextPage GetStructuredTextPage(int pageNumber, TesseractLanguage
01197            ocrLanguage, bool includeAnnotations = true, CancellationToken cancellationToken = default,
01198            IProgress<OCRProgressInfo> progress = null)
01199        {
01200            if (this.EncryptionState == EncryptionState.Encrypted)
01201            {
01202                throw new DocumentLockedException("A password is necessary to render the document!");
01203            }
01204
01205            if (DisplayLists[pageNumber] == null)
01206            {
01207                DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
01208                    this.Pages[pageNumber], includeAnnotations);
01209            }
01210        }
01211    }

```

```

01190         double zoom = 1;
01191         Rectangle region = this.Pages[pageNumber].Bounds;
01192
01193         if (this.ImageXRes != 72 || this.ImageYRes != 72)
01194         {
01195             zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
01196             region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 / 
01197             this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
01198         }
01199
01200         return new MuPDFStructuredTextPage(this.OwnerContext, this.DisplayLists[pageNumber],
01201             ocrLanguage, zoom, region, cancellationToken, progress);
01202     }
01203
01204     /// <summary>
01205     /// Creates a new <see cref="MuPDFStructuredTextPage"/> from the specified page, using optical
01206     /// character recognition (OCR) to determine what text is written on the image. This contains information
01207     /// about the text layout that can be used for highlighting and searching. The OCR step is run
01208     /// asynchronously, e.g. to avoid blocking the UI thread.
01209     /// </summary>
01210     /// <param name="pageNumber">The number of the page (starting at 0)</param>
01211     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
01212     /// this is null, no OCR is performed.</param>
01213     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01214     /// signatures) are included. Otherwise, only the page contents are included.</param>
01215     /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
01216     /// operation. Providing a value other than the default is not supported on Windows x86 and will throw a
01217     /// runtime exception.</param>
01218     /// <param name="progress">An <see cref="IProgress<OCRProgressInfo>"/> used to report
01219     /// progress. Providing a value other than null is not supported on Windows x86 and will throw a runtime
01220     /// exception.</param>
01221     /// <returns>A <see cref="MuPDFStructuredTextPage"/> containing a structured text
01222     /// representation of the page.</returns>
01223     public async Task<MuPDFStructuredTextPage> GetStructuredTextPageAsync(int pageNumber,
01224         TesseractLanguage ocrLanguage, bool includeAnnotations = true, CancellationToken cancellationToken =
01225             default, IProgress<OCRProgressInfo> progress = null)
01226     {
01227         if (this.EncryptionState == EncryptionState.Encrypted)
01228         {
01229             throw new DocumentLockedException("A password is necessary to render the document!");
01230         }
01231
01232         if (DisplayLists[pageNumber] == null)
01233         {
01234             DisplayLists[pageNumber] = new MuPDFDisplayList(this.OwnerContext,
01235                 this.Pages[pageNumber], includeAnnotations);
01236         }
01237
01238         double zoom = 1;
01239         Rectangle region = this.Pages[pageNumber].Bounds;
01240
01241         if (this.ImageXRes != 72 || this.ImageYRes != 72)
01242         {
01243             zoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
01244             region = new Rectangle(region.X0 * 72 / this.ImageXRes, region.Y0 * 72 / 
01245             this.ImageYRes, region.X1 * 72 / this.ImageXRes, region.Y1 * 72 / this.ImageYRes);
01246         }
01247
01248         return await Task.Run(() => new MuPDFStructuredTextPage(this.OwnerContext,
01249             this.DisplayLists[pageNumber], ocrLanguage, zoom, region, cancellationToken, progress));
01250     }
01251
01252     /// <summary>
01253     /// Extracts all the text from the document and returns it as a <see cref="string"/>. The
01254     /// reading order is taken from the order the text is drawn in the source file, so may not be accurate.
01255     /// </summary>
01256     /// <param name="separator">The character(s) used to separate the text lines obtained from the
01257     /// document. If this is <see langword="null" />, <see cref="Environment.NewLine"/> is used as a default
01258     /// separator.</param>
01259     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01260     /// signatures) are included. Otherwise, only the page contents are included.</param>
01261     /// <returns>A <see cref="string"/> containing all the text in the document. Characters are
01262     /// converted from the UTF-8 representation used in the document to equivalent UTF-16 <see
01263     /// cref="string"/>/s.</returns>
01264     public string ExtractText(string separator = null, bool includeAnnotations = true)
01265     {
01266         if (this.EncryptionState == EncryptionState.Encrypted)
01267         {
01268             throw new DocumentLockedException("A password is necessary to render the document!");
01269         }
01270
01271         separator = separator ?? Environment.NewLine;
01272
01273         var text = new StringBuilder();
01274         bool started = false;
01275
01276         for (int i = 0; i < this.Pages.Count; i++)
01277         {
01278             var page = this.Pages[i];
01279
01280             if (page.Text != null)
01281             {
01282                 text.Append(page.Text);
01283             }
01284         }
01285
01286         return text.ToString();
01287     }

```

```

01254         {
01255             MuPDFStructuredTextPage structuredTextPage = this.GetStructuredTextPage(i,
01256             includeAnnotations);
01257             foreach (MuPDFStructuredTextBlock textBlock in
01258                 structuredTextPage.StructuredTextBlocks)
01259             {
01260                 var numLines = textBlock.Count;
01261                 for (var j = 0; j < numLines; j++)
01262                 {
01263                     if (!string.IsNullOrWhiteSpace(textBlock[j].Text))
01264                     {
01265                         if (started)
01266                         {
01267                             text.Append(separator);
01268                         }
01269                         else
01270                         {
01271                             started = true;
01272                         }
01273                     text.Append(textBlock[j].Text);
01274                 }
01275             }
01276         }
01277     }
01278     return text.ToString();
01279 }
01280
01281 /// <summary>
01282     /// Extracts all the text from the document and returns it as a <see cref="string"/>, using
01283     /// optical character recognition (OCR) to determine what text is written on the image.
01284     /// </summary>
01285     /// <param name="separator">The character(s) used to separate the text lines obtained from the
01286     /// document. If this is <see langword="null" />, <see cref="Environment.NewLine"/> is used as a default
01287     /// separator.</param>
01288     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
01289     /// this is null, no OCR is performed.</param>
01290     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01291     /// signatures) are included. Otherwise, only the page contents are included.</param>
01292     /// <returns>A <see cref="string"/> containing all the text in the document. Characters are
01293     /// converted from the UTF-8 representation used in the document to equivalent UTF-16 <see
01294     /// cref="string"/>s.</returns>
01295     public string ExtractText(TesseractLanguage ocrLanguage, string separator = null, bool
01296     includeAnnotations = true)
01297     {
01298         if (this.EncryptionState == EncryptionState.Encrypted)
01299         {
01300             throw new DocumentLockedException("A password is necessary to render the document!");
01301         }
01302         separator = separator ?? Environment.NewLine;
01303
01304         var text = new StringBuilder();
01305         bool started = false;
01306
01307         for (int i = 0; i < this.Pages.Count; i++)
01308         {
01309             MuPDFStructuredTextPage structuredTextPage = this.GetStructuredTextPage(i,
01310             ocrLanguage, includeAnnotations);
01311             foreach (MuPDFStructuredTextBlock textBlock in
01312                 structuredTextPage.StructuredTextBlocks)
01313             {
01314                 var numLines = textBlock.Count;
01315                 for (var j = 0; j < numLines; j++)
01316                 {
01317                     if (!string.IsNullOrWhiteSpace(textBlock[j].Text))
01318                     {
01319                         if (started)
01320                         {
01321                             text.Append(separator);
01322                         }
01323                         else
01324                         {
01325                             started = true;
01326                         }
01327                     text.Append(textBlock[j].Text);
01328                 }
01329             }
01330         }
01331     }
01332     return text.ToString();
01333 }
01334
01335 /// <summary>

```

```

01329     /// Extracts all the text from the document and returns it as a <see cref="string"/>, using
01330     /// optical character recognition (OCR) to determine what text is written on the image. The OCR step is
01331     /// run asynchronously, e.g. to avoid blocking the UI thread.
01332     /// </summary>
01333     /// <param name="separator">The character(s) used to separate the text lines obtained from the
01334     /// document. If this is <see langword="null" />, <see cref="Environment.NewLine"/> is used as a default
01335     /// separator.</param>
01336     /// <param name="ocrLanguage">The language to use for optical character recognition (OCR). If
01337     /// this is null, no OCR is performed.</param>
01338     /// <param name="includeAnnotations">If this is <see langword="true" />, annotations (e.g.
01339     /// signatures) are included. Otherwise, only the page contents are included.</param>
01340     /// <param name="cancellationToken">A <see cref="CancellationToken"/> used to cancel the
01341     /// operation.</param>
01342     /// <param name="progress">An <see cref="IProgress<OCRProgressInfo>"/> used to report
01343     /// progress.</param>
01344     /// <returns>A <see cref="string"/> containing all the text in the document. Characters are
01345     /// converted from the UTF-8 representation used in the document to equivalent UTF-16 <see
01346     /// cref="string"/>s.</returns>
01347     public async Task<string> ExtractTextAsync(TesseractLanguage ocrLanguage, string separator =
01348         null, bool includeAnnotations = true, CancellationToken cancellationToken = default,
01349         IProgress<OCRProgressInfo> progress = null)
01350     {
01351         if (this.EncryptionState == EncryptionState.Encrypted)
01352         {
01353             throw new DocumentLockedException("A password is necessary to render the document!");
01354         }
01355         separator = separator ?? Environment.NewLine;
01356         var text = new StringBuilder();
01357         bool started = false;
01358         for (int i = 0; i < this.Pages.Count; i++)
01359         {
01360             MuPDFStructuredTextPage structuredTextPage = await this.GetStructuredTextPageAsync(i,
01361                 ocrLanguage, includeAnnotations, cancellationToken, progress);
01362             foreach (MuPDFstructuredTextBlock textBlock in
01363                 structuredTextPage.StructuredTextBlocks)
01364             {
01365                 var numLines = textBlock.Count;
01366                 for (var j = 0; j < numLines; j++)
01367                 {
01368                     if (!string.IsNullOrWhiteSpace(textBlock[j].Text))
01369                     {
01370                         if (started)
01371                         {
01372                             text.Append(separator);
01373                         }
01374                         return text.ToString();
01375                     }
01376                 }
01377             }
01378             /// <summary>
01379             /// Attempts to unlock the document with the supplied password.
01380             /// </summary>
01381             /// <param name="password">The user or owner password to use to unlock the document.</param>
01382             /// <returns><see langword="true"/> if the document was successfully unlocked (or if it was
01383             /// never locked to begin with), <see langword="false"/> if the password was incorrect and the document
01384             /// is still locked.</returns>
01385             /// <remarks>This method can be used both to unlock an encrypted document and to check whether
01386             /// the supplied owner password is correct.</remarks>
01387             public bool TryUnlock(string password)
01388             {
01389                 return TryUnlock(password, out _);
01390             }
01391             /// <summary>
01392             /// Attempts to unlock the document with the supplied password.
01393             /// </summary>
01394             /// <param name="password">The user or owner password to use to unlock the document.</param>
01395             /// <param name="passwordType">If the method returns true, this can be used to determine
01396             /// whether the supplied password was the user password or the owner password. If the method returns <see
01397             /// langword="false"/>,
01398             /// this can be used to determine whether a user password and/or an owner password are
01399             /// required.</param>
01400             /// <returns><see langword="true"/> if the document was successfully unlocked (or if it was
01401             /// never locked to begin with), <see langword="false"/> if the password was incorrect and the document

```

```
    is still locked.</returns>
01395     /// <remarks>This method can be used both to unlock an encrypted document and to check whether
01396     // the supplied owner password is correct.</remarks>
01397     public bool TryUnlock(string password, out PasswordTypes passwordType)
01398     {
01399         int result = NativeMethods.UnlockWithPassword(this.OwnerContext.NativeContext,
01400             this.NativeDocument, password);
01401
01402         int pt = 0;
01403
01404         switch (result)
01405         {
01406             case 0:
01407                 pt = 0;
01408
01409                 if (this.EncryptionState == EncryptionState.Encrypted)
01410                 {
01411                     pt |= 1;
01412
01413                     if (this.RestrictionState == RestrictionState.Restricted)
01414                     {
01415                         pt |= 2;
01416
01417                     passwordType = (PasswordTypes)pt;
01418                     return !(this.EncryptionState == EncryptionState.Encrypted ||
01419                         this.RestrictionState == RestrictionState.Restricted);
01420
01421                     case 1:
01422                         passwordType = PasswordTypes.None;
01423                         return true;
01424
01425                     case 2:
01426                         if (this.EncryptionState == EncryptionState.Encrypted)
01427                         {
01428                             this.EncryptionState = EncryptionState.Unlocked;
01429
01430                         passwordType = PasswordTypes.User;
01431                         return true;
01432
01433                     case 4:
01434                         if (this.RestrictionState == RestrictionState.Restricted)
01435                         {
01436                             this.RestrictionState = RestrictionState.Unlocked;
01437
01438                         passwordType = PasswordTypes.Owner;
01439                         return true;
01440
01441                     case 6:
01442                         pt = 0;
01443
01444                         if (this.EncryptionState == EncryptionState.Encrypted)
01445                         {
01446                             this.EncryptionState = EncryptionState.Unlocked;
01447                             pt |= 1;
01448
01449                         if (this.RestrictionState == RestrictionState.Restricted)
01450                         {
01451                             this.RestrictionState = RestrictionState.Unlocked;
01452                             pt |= 2;
01453
01454                         passwordType = (PasswordTypes)pt;
01455                         return true;
01456
01457                     default:
01458                         throw new ArgumentOutOfRangeException("Unexpected return value when unlocking the
01459                         document: " + result.ToString());
01460
01461
01462         private bool disposedValue;
01463
01464         //<inheritDoc/>
01465         protected virtual void Dispose(bool disposing)
01466         {
01467             if (!disposedValue)
01468             {
01469                 if (disposing)
01470                 {
01471                     Pages.Dispose();
01472                     foreach (MuPDFDisplayList list in DisplayLists)
01473                     {
01474                         list?.Dispose();
01475
01476                         DataHandle?.Free();
01477                         DataHolder?.Dispose();
01478
01479                     }
01480
01481                     NativeMethods.DisposeDocument(OwnerContext.NativeContext, NativeDocument);
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02509
02510
02511
02512
02513
02514
02515
02516
02517
02518
02519
02519
02520
02521
02522
02523
02524
02525
02526
02527
02528
02529
02529
02530
02531
02532
02533
02534
02535
02536
02537
02538
02539
02539
02540
02541
02542
02543
02544
02545
02546
02547
02548
02549
02549
02550
02551
02552
02553
02554
02555
02556
02557
02558
02559
02559
02560
02561
02562
02563
02564
02565
02566
02567
02568
02569
02569
02570
02571
02572
02573
02574
02575
02576
02577
02578
02579
02579
02580
02581
02582
02583
02584
02585
02586
02587
02588
02589
02589
02590
02591
02592
02593
02594
02595
02596
02597
02598
02599
02599
02600
02601
02602
02603
02604
02605
02606
02607
02608
02609
02609
02610
02611
02612
02613
02614
02615
02616
02617
02618
02619
02619
02620
02621
02622
02623
02624
02625
02626
02627
02628
02629
02629
02630
02631
02632
02633
02634
02635
02636
02637
02638
02639
02639
02640
02641
02642
02643
02644
02645
02646
02647
02648
02649
02649
02650
02651
02652
02653
02654
02655
02656
02657
02658
02659
02659
02660
02661
02662
02663
02664
02665
02666
02667
02668
02669
02669
02670
02671
02672
02673
02674
02675
02676
02677
02678
02679
02679
02680
02681
02682
02683
02684
02685
02686
02687
02688
02689
02689
02690
02691
02692
02693
02694
02695
02696
02697
02698
02699
02699
02700
02701
02702
02703
02704
02705
02706
02707
02708
02709
02709
02710
02711
02712
02713
02714
02715
02716
02717
02718
02719
02719
02720
02721
02722
02723
02724
02725
02726
02727
02728
02729
02729
02730
02731
02732
02733
02734
02735
02736
02737
02738
02739
02739
02740
02741
02742
02743
02744
02745
02746
02747
02748
02749
02749
02750
02751
02752
02753
02754
02755
02756
02757
02758
02759
02759
02760
02761
02762
02763
02764
02765
02766
02767
02768
02769
02769
02770
02771
02772
02773
02774
02775
02776
02777
02778
02779
02779
02780
02781
02782
02783
02784
02785
02786
02787
02788
02789
02789
02790
02791
02792
02793
02794
02795
02796
02797
02798
02799
02799
02800
02801
02802
02803
02804
02805
02806
02807
02808
02809
02809
02810
02811
02812
02813
02814
02815
02816
02817
02818
02819
02819
02820
02821
02822
02823
02824
02825
02826
02827
02828
02829
02829
02830
02831
02832
02833
02834
02835
02836
02837
02838
02839
02839
02840
02841
02842
02843
02844
02845
02846
02847
02848
02849
02849
02850
02851
02852
02853
02854
02855
02856
02857
02858
02859
02859
02860
02861
02862
02863
02864
02865
02866
02867
02868
02869
02869
02870
02871
02872
02873
02874
02875
02876
02877
02878
02879
02879
02880
02881
02882
02883
02884
02885
02886
02887
02888
02889
02889
02890
02891
02892
02893
02894
02895
02896
02897
02898
02899
02899
02900
02901
02902
02903
02904
02905
02906
02907
02908
02909
02909
02910
02911
02912
02913
02914
02915
02916
02917
02918
02919
02919
02920
02921
02922
02923
02924
02925
02926
02927
02928
02929
02929
02930
02931
02932
02933
02934
02935
02936
02937
02938
02939
02939
02940
02941
02942
02943
02944
02945
02946
02947
02948
02949
02949
02950
02951
02952
02953
02954
02955
02956
02957
02958
02959
02959
02960
02961
02962
02963
02964
02965
02966
02967
02968
02969
02969
02970
02971
02972
02973
02974
02975
02976
02977
02978
02979
02979
02980
02981
02982
02983
02984
02985
02986
02987
02988
02989
02989
02990
02991
02992
02993
02994
02995
02996
02997
02998
02999
02999
03000
03001
03002
03003
03004
03005
03006
03007
03008
03009
03009
03010
03011
03012
03013
03014
03015
03016
03017
03018
03019
03019
03020
03021
03022
03023
03024
03025
03026
03027
03028
03029
03029
03030
03031
03032
03033
03034
03035
03036
03037
03038
03039
03039
03040
03041
03042
03043
03044
03045
03046
03047
03048
03049
03049
03050
03051
03052
03053
03054
03055
03056
03057
03058
03059
03059
03060
03061
03062
0306
```

```

01477             if (NativeStream != IntPtr.Zero)
01478             {
01479                 NativeMethods.DisposeStream(OwnerContext.NativeContext, NativeStream);
01480             }
01481         }
01482         disposedValue = true;
01483     }
01484 }
01485 }
01486
01487 ///<inheritdoc/>
01488 ~MuPDFDocument()
01489 {
01490     Dispose(disposing: false);
01491 }
01492
01493 ///<inheritdoc/>
01494 public void Dispose()
01495 {
01496     Dispose(disposing: true);
01497     GC.SuppressFinalize(this);
01498 }
01499 }
01500 }

```

8.8 MuPDFMultiThreadedPageRenderer.cs

```

00001 /*
00002  MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
00003  Copyright (C) 2020 Giorgio Bianchini
00004
00005  This program is free software: you can redistribute it and/or modify
00006  it under the terms of the GNU Affero General Public License as
00007  published by the Free Software Foundation, version 3.
00008
00009  This program is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU Affero General Public License for more details.
00013
00014  You should have received a copy of the GNU Affero General Public License
00015  along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using System;
00019 using System.Collections.Generic;
00020 using System.Linq;
00021 using System.Runtime.InteropServices;
00022 using System.Threading;
00023
00024 namespace MuPDFCore
00025 {
00026     /// <summary>
00027     /// A reusable thread to render part of an image.
00028     /// </summary>
00029     internal class RenderingThread : IDisposable
00030     {
00031         /// <summary>
00032         /// A class to hold the data used internally by the rendering thread.
00033         /// </summary>
00034         private class RenderData
00035         {
00036             public IntPtr Context;
00037             public MuPDFDisplayList DisplayList;
00038             public Rectangle Region;
00039             public float Zoom;
00040             public IntPtr PixelStorage;
00041             public Rectangle PageBounds;
00042             public PixelFormats PixelFormat;
00043             public bool ClipToPageBounds;
00044         }
00045
00046         /// <summary>
00047         /// A lock object to prevent race conditions.
00048         /// </summary>
00049         private readonly object RenderDataLock;
00050
00051         /// <summary>
00052         /// The data used internally by the rendering thread.
00053         /// </summary>
00054         private readonly RenderData CurrentRenderData;
00055
00056         /// <summary>

```

```
00057     /// The actual thread that does the rendering.  
00058     /// </summary>  
00059     private readonly Thread Thread;  
00060  
00061     /// <summary>  
00062     /// An <see cref="EventWaitHandle"/> that is set by the <see cref="Thread"/> when it finished  
00063     /// rendering.  
00064     private readonly EventWaitHandle SignalFromThread;  
00065  
00066     /// <summary>  
00067     /// An <see cref="EventWaitHandle"/> that signals to the <see cref="Thread"/> that it should  
00068     start rendering.  
00069     private readonly EventWaitHandle SignalToThread;  
00070  
00071     /// <summary>  
00072     /// An <see cref="EventWaitHandle"/> that signals that the object is being disposed and all  
00073     activity should cease.  
00074     private readonly EventWaitHandle DisposeSignal;  
00075  
00076     /// <summary>  
00077     /// A pointer to a <see cref="Cookie"/> object that can be used to monitor the progress of the  
00078     rendering or to abort it.  
00079     private readonly IntPtr Cookie;  
00080  
00081     /// <summary>  
00082     /// Performs the actual rendering.  
00083     /// </summary>  
00084     private void RenderAction()  
00085     {  
00086         ExitCodes result =  
00087         (ExitCodes)NativeMethods.RenderSubDisplayList(this.CurrentRenderData.Context,  
this.CurrentRenderData.DisplayList.NativeDisplayList, this.CurrentRenderData.Region.X0,  
this.CurrentRenderData.Region.Y0, this.CurrentRenderData.Region.X1, this.CurrentRenderData.Region.Y1,  
this.CurrentRenderData.Zoom, (int)this.CurrentRenderData.PixelFormat,  
this.CurrentRenderData.PixelStorage, Cookie);  
00088         switch (result)  
00089         {  
00090             case ExitCodes.EXIT_SUCCESS:  
00091                 break;  
00092             case ExitCodes.ERR_CANNOT_RENDER:  
00093                 throw new MuPDFException("Cannot render page", result);  
00094             default:  
00095                 throw new MuPDFException("Unknown error", result);  
00096         }  
00097  
00098         RoundedRectangle roundedRegion =  
this.CurrentRenderData.Region.Round(this.CurrentRenderData.Zoom);  
00099         RoundedSize roundedSize = new RoundedSize(roundedRegion.Width, roundedRegion.Height);  
00100  
00101         if (this.CurrentRenderData.PixelFormat == PixelFormats.RGBA ||  
this.CurrentRenderData.PixelFormat == PixelFormats.BGRA)  
00102         {  
00103             Utils.UnpremultiplyAlpha(this.CurrentRenderData.PixelStorage, roundedSize);  
00104         }  
00105  
00106         if (this.CurrentRenderData.ClipToPageBounds &&  
!this.CurrentRenderData.PageBounds.Contains(this.CurrentRenderData.DisplayList.Bounds.Intersect(this.CurrentRenderData.  
00107         {  
00108             Utils.ClipImage(this.CurrentRenderData.PixelStorage, roundedSize,  
this.CurrentRenderData.Region, this.CurrentRenderData.PageBounds,  
this.CurrentRenderData.PixelFormat);  
00109         }  
00110     }  
00111  
00112     /// <summary>  
00113     /// Create a new <see cref="RenderingThread"/> instance.  
00114     /// </summary>  
00115     public RenderingThread()  
00116     {  
00117         //Initialize fields  
00118         SignalFromThread = new EventWaitHandle(false, EventResetMode.ManualReset);  
00119         SignalToThread = new EventWaitHandle(false, EventResetMode.ManualReset);  
00120         DisposeSignal = new EventWaitHandle(false, EventResetMode.ManualReset);  
00121  
00122         CurrentRenderData = new RenderData();  
00123         RenderDataLock = new object();  
00124  
00125         //Allocate unmanaged memory to hold the cookie.  
00126         Cookie = Marshal.AllocHGlobal(Marshal.SizeOf<MuPDFCore.Cookie>());  
00127  
00128         //Initialise the rendering thread.  
00129         this.Thread = new Thread(() =>
```

```

00130         {
00131             EventWaitHandle[] handles = new EventWaitHandle[] { SignalToThread, DisposeSignal };
00132
00133             while (true)
00134             {
00135                 int handle = EventWaitHandle.WaitAny(handles);
00136
00137                 if (handle == 0)
00138                 {
00139                     SignalToThread.Reset();
00140
00141                     lock (RenderDataLock)
00142                     {
00143                         this.RenderAction();
00144                     }
00145
00146                     SignalFromThread.Set();
00147                 }
00148                 else
00149                 {
00150                     SignalFromThread.Set();
00151                     break;
00152                 }
00153             }
00154         });
00155
00156         //Start the rendering thread.
00157         this.Thread.Start();
00158     }
00159
00160     /// <summary>
00161     /// Start rendering a region of a display list to the specified destination.
00162     /// </summary>
00163     /// <param name="context">The rendering context.</param>
00164     /// <param name="displayList">The native display list that should be rendered.</param>
00165     /// <param name="region">The region that should be rendered.</param>
00166     /// <param name="zoom">The scale at which the region will be rendered. This will determine the
00167     /// size in pixel of the image.</param>
00168     /// <param name="pixelStorage">The address of the buffer where the pixel data will be written.
00169     /// There must be enough space available to write the values for all the pixels, otherwise this will fail
00170     /// catastrophically!</param>
00171     /// <param name="pixelFormat">The format of the pixel data.</param>
00172     /// <param name="pageBounds">The bounds of the page being rendered.</param>
00173     /// <param name="clipToPageBounds">A boolean value indicating whether the rendered image
00174     /// should be clipped to the original page's bounds. This can be relevant if the page has been "cropped"
00175     /// by altering its mediabox, but otherwise leaving the contents untouched.</param>
00176     public void Render(IntPtr context, MuPDFDisplayList displayList, Rectangle region, float zoom,
00177     IntPtr pixelStorage, PixelFormats pixelFormat, Rectangle pageBounds, bool clipToPageBounds)
00178     {
00179         lock (RenderDataLock)
00180         {
00181             //Reset the cookie.
00182             unsafe
00183             {
00184                 Cookie* cookie = (Cookie*)Cookie;
00185
00186                 cookie->abort = 0;
00187                 cookie->errors = 0;
00188                 cookie->incomplete = 0;
00189                 cookie->progress = 0;
00190                 cookie->progress_max = 0;
00191
00192             //Set up all the rendering data.
00193             CurrentRenderData.Context = context;
00194             CurrentRenderData.DisplayList = displayList;
00195             CurrentRenderData.Region = region;
00196             CurrentRenderData.Zoom = zoom;
00197             CurrentRenderData.PixelStorage = pixelStorage;
00198             CurrentRenderData.PixelFormat = pixelFormat;
00199             CurrentRenderData.PageBounds = pageBounds;
00200             CurrentRenderData.ClipToPageBounds = clipToPageBounds;
00201             SignalToThread.Set();
00202         }
00203
00204     /// <summary>
00205     /// Wait until the current rendering operation finishes.
00206     /// </summary>
00207     public void WaitForRendering()
00208     {
00209         EventWaitHandle[] handles = new EventWaitHandle[] { SignalFromThread, DisposeSignal };
00210         int result = EventWaitHandle.WaitAny(handles);
00211         if (result == 0)
00212         {
00213             SignalFromThread.Reset();
00214         }
00215     }

```

```

00211         }
00212
00213         /// <summary>
00214         /// Abort the current rendering operation.
00215         /// </summary>
00216         public void AbortRendering()
00217     {
00218         lock (RenderDataLock)
00219     {
00220         unsafe
00221     {
00222         Cookie* cookie = (Cookie*)Cookie;
00223         cookie->abort = 1;
00224     }
00225 }
00226 }
00227
00228         /// <summary>
00229         /// Get the progress of the current rendering operation.
00230         /// </summary>
00231         /// <returns>A <see cref="RenderProgress.ThreadRenderProgress"/> object containing the
00232         /// progress of the current rendering operation.</returns>
00233         public RenderProgress.ThreadRenderProgress GetProgress()
00234     {
00235         int progress;
00236         ulong maxProgress;
00237
00238         unsafe
00239     {
00240         Cookie* cookie = (Cookie*)Cookie;
00241
00242         progress = cookie->progress;
00243         maxProgress = cookie->progress_max;
00244     }
00245
00246         return new RenderProgress.ThreadRenderProgress(progress, maxProgress);
00247     }
00248
00249     private bool disposedValue;
00250
00251     protected virtual void Dispose(bool disposing)
00252     {
00253         if (!disposedValue)
00254     {
00255         if (disposing)
00256     {
00257         DisposeSignal.Set();
00258         Thread.Join();
00259     }
00260
00261         Marshal.FreeHGlobal(Cookie);
00262
00263         disposedValue = true;
00264     }
00265
00266     public void Dispose()
00267     {
00268         Dispose(disposing: true);
00269         GC.SuppressFinalize(this);
00270     }
00271 }
00272
00273         /// <summary>
00274         /// A class that holds the necessary resources to render a page of a MuPDF document using multiple
00275         /// threads.
00276         /// </summary>
00277         public class MuPDFMultiThreadedPageRenderer : IDisposable
00278     {
00279         /// <summary>
00280         /// The display list that is rendered by this renderer.
00281         /// </summary>
00282         private readonly MuPDFDisplayList DisplayList;
00283
00284         /// <summary>
00285         /// The cloned contexts that are used by the <see cref="RenderingThreads"/> to render the
00286         /// display list.
00287         /// </summary>
00288         private readonly MuPDFContext[] Contexts;
00289
00290         /// <summary>
00291         /// The <see cref="RenderingThreads"/> that are in charge of the actual rendering.
00292         /// </summary>
00293         private readonly RenderingThread[] RenderingThreads;
00294
00295         /// <summary>
00296         /// The bounds of the page being rendered.
00297     }

```

```
00295
00296     /// </summary>
00297     private readonly Rectangle PageBounds;
00298
00299     /// <summary>
00300     /// Whether the rendered images should be clipped to the bounds of the page being rendered.
00301     /// </summary>
00302     private readonly bool ClipToPageBounds;
00303
00304     /// <summary>
00305     /// The number of threads that are used to render the image.
00306     /// </summary>
00307     public int ThreadCount { get; }
00308
00309     /// <summary>
00310     /// If the document is an image, the horizontal resolution of the image. Otherwise, 72.
00311     /// </summary>
00312     internal double ImageXRes = double.NaN;
00313
00314     /// <summary>
00315     /// If the document is an image, the vertical resolution of the image. Otherwise, 72.
00316     internal double ImageYRes = double.NaN;
00317
00318     /// <summary>
00319     /// Create a new <see cref="MuPDFMultiThreadedPageRenderer"/> from a specified display list
00320     /// using the specified number of threads.
00321     /// </summary>
00322     /// <param name="context">The context that owns the document from which the display list was
00323     /// extracted.</param>
00324     /// <param name="displayList">The display list to render.</param>
00325     /// <param name="threadCount">The number of threads to use in the rendering. This must be
00326     /// factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than <paramref
00327     /// name="threadCount"/> that satisfies this condition is used.</param>
00328     /// <param name="pageBounds">The bounds of the page being rendered.</param>
00329     /// <param name="clipToPageBounds">A boolean value indicating whether the rendered image
00330     /// should be clipped to the original page's bounds. This can be relevant if the page has been "cropped"
00331     /// by altering its mediabox, but otherwise leaving the contents untouched.</param>
00332     /// <param name="imageXRes">If the document is an image, the horizontal resolution of the
00333     /// image. Otherwise, 72.</param>
00334     /// <param name="imageYRes">If the document is an image, the vertical resolution of the image.
00335     /// Otherwise, 72.</param>
00336     internal MuPDFMultiThreadedPageRenderer(MuPDFContext context, MuPDFDisplayList displayList,
00337     int threadCount, Rectangle pageBounds, bool clipToPageBounds, double imageXRes, double imageYRes)
00338     {
00339         threadCount = Utils.GetAcceptableNumber(threadCount);
00340
00341         this.ThreadCount = threadCount;
00342         this.DisplayList = displayList;
00343         this.PageBounds = pageBounds;
00344         this.ClipToPageBounds = clipToPageBounds;
00345
00346         this.ImageXRes = imageXRes;
00347         this.ImageYRes = imageYRes;
00348
00349         IntPtr[] contexts = new IntPtr[threadCount];
00350         GCHandle contextsHandle = GCHandle.Alloc(contexts, GCHandleType.Pinned);
00351
00352         try
00353         {
00354             ExitCodes result = (ExitCodes)NativeMethods.CloneContext(context.NativeContext,
00355             threadCount, contextsHandle.AddrOfPinnedObject());
00356
00357             switch (result)
00358             {
00359                 case ExitCodes.EXIT_SUCCESS:
00360                     break;
00361                 case ExitCodes.ERR_CANNOT_INIT_MUTEX:
00362                     throw new MuPDFException("Cannot initialize mutex objects", result);
00363                 case ExitCodes.ERR_CANNOT_CREATE_CONTEXT:
00364                     throw new MuPDFException("Cannot create master context", result);
00365                 case ExitCodes.ERR_CANNOT_CLONE_CONTEXT:
00366                     throw new MuPDFException("Cannot create context clones", result);
00367                 default:
00368                     throw new MuPDFException("Unknown error", result);
00369             }
00370
00371             Contexts = new MuPDFContext[threadCount];
00372             RenderingThreads = new RenderingThread[threadCount];
00373             for (int i = 0; i < threadCount; i++)
00374             {
00375                 Contexts[i] = new MuPDFContext(contexts[i]);
00376                 RenderingThreads[i] = new RenderingThread();
00377             }
00378         }
00379         finally
00380         {
00381             contextsHandle.Free();
00382         }
00383     }
00384 }
```

```

00372         }
00373     }
00374
00375     /// <summary>
00376     /// Render the specified region to an image of the specified size, split in a number of tiles
00377     /// equal to the number of threads used by this <see cref="MuPDFMultiThreadedPageRenderer"/>, without
00378     /// marshaling. This method will not return until all the rendering threads have finished.
00379     /// </summary>
00380     /// <param name="targetSize">The total size of the image that should be rendered.</param>
00381     /// <param name="region">The region in page units that should be rendered.</param>
00382     /// <param name="destinations">An array containing the addresses of the buffers where the
00383     /// rendered tiles will be written. There must be enough space available in each buffer to write the
00384     /// values for all the pixels of the tile, otherwise this will fail catastrophically!
00385     /// As long as the <paramref name="targetSize"/> is the same, the size in pixel of the tiles
00386     /// is guaranteed to also be the same.</param>
00387     /// <param name="pixelFormat">The format of the pixel data.</param>
00388     public void Render(RoundedSize targetSize, Rectangle region, IntPtr[] destinations,
00389     PixelFormats pixelFormat)
00390     {
00391         if (destinations.Length != Contexts.Length)
00392         {
00393             throw new ArgumentOutOfRangeException(nameof(destinations), destinations.Length, "The
00394             number of destinations must be equal to the number of rendering threads!");
00395         }
00396
00397         RoundedRectangle[] targets = targetSize.Split(destinations.Length);
00398
00399         float zoomX = targetSize.Width / region.Width;
00400         float zoomY = targetSize.Height / region.Height;
00401
00402         float zoom = (float)Math.Sqrt(zoomX * zoomY);
00403
00404         Rectangle actualPageArea = new Rectangle(region.X0, region.Y0, region.X0 +
00405         targetSize.Width / zoom, region.Y0 + targetSize.Height / zoom);
00406
00407         Rectangle[] origins = actualPageArea.Split(destinations.Length);
00408
00409         //Make sure that each tile has the expected size in pixel, rounding errors
00410         notwithstanding.
00411         for (int i = 0; i < origins.Length; i++)
00412         {
00413             int countBlanks = 0;
00414             RoundedRectangle roundedOrigin = origins[i].Round(zoom);
00415             while (roundedOrigin.Width != targets[i].Width || roundedOrigin.Height !=
00416             targets[i].Height)
00417             {
00418                 RoundedRectangle oldRoundedOrigin = roundedOrigin;
00419
00420                 if (roundedOrigin.Width > targets[i].Width)
00421                 {
00422                     if (origins[i].X0 > actualPageArea.X0)
00423                     {
00424                         origins[i] = new Rectangle(origins[i].X0 + 0.5 / zoom, origins[i].Y0,
00425                         origins[i].X1, origins[i].Y1);
00426                     }
00427                     else
00428                     {
00429                         origins[i] = new Rectangle(origins[i].X0, origins[i].Y0, origins[i].X1 -
00430                         0.5 / zoom, origins[i].Y1);
00431                     }
00432
00433                 if (roundedOrigin.Height > targets[i].Height)
00434                 {
00435                     if (origins[i].Y0 > actualPageArea.Y0)
00436                     {
00437                         origins[i] = new Rectangle(origins[i].X0, origins[i].Y0 + 0.5 / zoom,
00438                         origins[i].X1, origins[i].Y1);
00439                     }
00440                     else
00441                     {
00442                         origins[i] = new Rectangle(origins[i].X0, origins[i].Y0, origins[i].X1,
00443                         origins[i].Y1 - 0.5 / zoom);
00444
00445                 }
00446             }
00447         }
00448     }
00449
00450     /// <summary>
00451     /// Get the current rendering context.
00452     /// </summary>
00453     /// <returns>The current rendering context.</returns&gt;
00454     public Context Context { get; private set; }
00455
00456     /// &lt;summary&gt;
00457     /// Set the current rendering context.
00458     /// &lt;/summary&gt;
00459     /// &lt;param name="context"&gt;The new rendering context.</param&gt;
00460     public void SetContext(Context context)
00461     {
00462         Context = context;
00463     }
00464
00465     /// &lt;summary&gt;
00466     /// Get the current rendering thread ID.
00467     /// &lt;/summary&gt;
00468     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00469     public int ThreadId { get; private set; }
00470
00471     /// &lt;summary&gt;
00472     /// Set the current rendering thread ID.
00473     /// &lt;/summary&gt;
00474     /// &lt;param name="threadId"&gt;The new rendering thread ID.</param&gt;
00475     public void SetThreadId(int threadId)
00476     {
00477         ThreadId = threadId;
00478     }
00479
00480     /// &lt;summary&gt;
00481     /// Get the current rendering thread ID.
00482     /// &lt;/summary&gt;
00483     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00484     public int CurrentThreadId { get; private set; }
00485
00486     /// &lt;summary&gt;
00487     /// Set the current rendering thread ID.
00488     /// &lt;/summary&gt;
00489     /// &lt;param name="currentThreadId"&gt;The new rendering thread ID.</param&gt;
00490     public void SetCurrentThreadId(int currentThreadId)
00491     {
00492         CurrentThreadId = currentThreadId;
00493     }
00494
00495     /// &lt;summary&gt;
00496     /// Get the current rendering context.
00497     /// &lt;/summary&gt;
00498     /// &lt;returns&gt;The current rendering context.</returns&gt;
00499     public Context GetCurrentContext()
00500     {
00501         return Context;
00502     }
00503
00504     /// &lt;summary&gt;
00505     /// Set the current rendering context.
00506     /// &lt;/summary&gt;
00507     /// &lt;param name="currentContext"&gt;The new rendering context.</param&gt;
00508     public void SetCurrentContext(Context currentContext)
00509     {
00510         Context = currentContext;
00511     }
00512
00513     /// &lt;summary&gt;
00514     /// Get the current rendering thread ID.
00515     /// &lt;/summary&gt;
00516     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00517     public int GetCurrentThreadId()
00518     {
00519         return ThreadId;
00520     }
00521
00522     /// &lt;summary&gt;
00523     /// Set the current rendering thread ID.
00524     /// &lt;/summary&gt;
00525     /// &lt;param name="currentThreadId"&gt;The new rendering thread ID.</param&gt;
00526     public void SetCurrentThreadId(int currentThreadId)
00527     {
00528         ThreadId = currentThreadId;
00529     }
00530
00531     /// &lt;summary&gt;
00532     /// Get the current rendering context.
00533     /// &lt;/summary&gt;
00534     /// &lt;returns&gt;The current rendering context.</returns&gt;
00535     public Context GetDefaultContext()
00536     {
00537         return Context;
00538     }
00539
00540     /// &lt;summary&gt;
00541     /// Set the current rendering context.
00542     /// &lt;/summary&gt;
00543     /// &lt;param name="defaultContext"&gt;The new rendering context.</param&gt;
00544     public void SetDefaultContext(Context defaultContext)
00545     {
00546         Context = defaultContext;
00547     }
00548
00549     /// &lt;summary&gt;
00550     /// Get the current rendering thread ID.
00551     /// &lt;/summary&gt;
00552     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00553     public int GetDefaultThreadId()
00554     {
00555         return ThreadId;
00556     }
00557
00558     /// &lt;summary&gt;
00559     /// Set the current rendering thread ID.
00560     /// &lt;/summary&gt;
00561     /// &lt;param name="defaultThreadId"&gt;The new rendering thread ID.</param&gt;
00562     public void SetDefaultThreadId(int defaultThreadId)
00563     {
00564         ThreadId = defaultThreadId;
00565     }
00566
00567     /// &lt;summary&gt;
00568     /// Get the current rendering context.
00569     /// &lt;/summary&gt;
00570     /// &lt;returns&gt;The current rendering context.</returns&gt;
00571     public Context GetMainContext()
00572     {
00573         return Context;
00574     }
00575
00576     /// &lt;summary&gt;
00577     /// Set the current rendering context.
00578     /// &lt;/summary&gt;
00579     /// &lt;param name="mainContext"&gt;The new rendering context.</param&gt;
00580     public void SetMainContext(Context mainContext)
00581     {
00582         Context = mainContext;
00583     }
00584
00585     /// &lt;summary&gt;
00586     /// Get the current rendering thread ID.
00587     /// &lt;/summary&gt;
00588     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00589     public int GetMainThreadId()
00590     {
00591         return ThreadId;
00592     }
00593
00594     /// &lt;summary&gt;
00595     /// Set the current rendering thread ID.
00596     /// &lt;/summary&gt;
00597     /// &lt;param name="mainThreadId"&gt;The new rendering thread ID.</param&gt;
00598     public void SetMainThreadId(int mainThreadId)
00599     {
00600         ThreadId = mainThreadId;
00601     }
00602
00603     /// &lt;summary&gt;
00604     /// Get the current rendering context.
00605     /// &lt;/summary&gt;
00606     /// &lt;returns&gt;The current rendering context.</returns&gt;
00607     public Context GetSecondaryContext()
00608     {
00609         return Context;
00610     }
00611
00612     /// &lt;summary&gt;
00613     /// Set the current rendering context.
00614     /// &lt;/summary&gt;
00615     /// &lt;param name="secondaryContext"&gt;The new rendering context.</param&gt;
00616     public void SetSecondaryContext(Context secondaryContext)
00617     {
00618         Context = secondaryContext;
00619     }
00620
00621     /// &lt;summary&gt;
00622     /// Get the current rendering thread ID.
00623     /// &lt;/summary&gt;
00624     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00625     public int GetSecondaryThreadId()
00626     {
00627         return ThreadId;
00628     }
00629
00630     /// &lt;summary&gt;
00631     /// Set the current rendering thread ID.
00632     /// &lt;/summary&gt;
00633     /// &lt;param name="secondaryThreadId"&gt;The new rendering thread ID.</param&gt;
00634     public void SetSecondaryThreadId(int secondaryThreadId)
00635     {
00636         ThreadId = secondaryThreadId;
00637     }
00638
00639     /// &lt;summary&gt;
00640     /// Get the current rendering context.
00641     /// &lt;/summary&gt;
00642     /// &lt;returns&gt;The current rendering context.</returns&gt;
00643     public Context GetTertiaryContext()
00644     {
00645         return Context;
00646     }
00647
00648     /// &lt;summary&gt;
00649     /// Set the current rendering context.
00650     /// &lt;/summary&gt;
00651     /// &lt;param name="tertiaryContext"&gt;The new rendering context.</param&gt;
00652     public void SetTertiaryContext(Context tertiaryContext)
00653     {
00654         Context = tertiaryContext;
00655     }
00656
00657     /// &lt;summary&gt;
00658     /// Get the current rendering thread ID.
00659     /// &lt;/summary&gt;
00660     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00661     public int GetTertiaryThreadId()
00662     {
00663         return ThreadId;
00664     }
00665
00666     /// &lt;summary&gt;
00667     /// Set the current rendering thread ID.
00668     /// &lt;/summary&gt;
00669     /// &lt;param name="tertiaryThreadId"&gt;The new rendering thread ID.</param&gt;
00670     public void SetTertiaryThreadId(int tertiaryThreadId)
00671     {
00672         ThreadId = tertiaryThreadId;
00673     }
00674
00675     /// &lt;summary&gt;
00676     /// Get the current rendering context.
00677     /// &lt;/summary&gt;
00678     /// &lt;returns&gt;The current rendering context.</returns&gt;
00679     public Context GetQuaternaryContext()
00680     {
00681         return Context;
00682     }
00683
00684     /// &lt;summary&gt;
00685     /// Set the current rendering context.
00686     /// &lt;/summary&gt;
00687     /// &lt;param name="quaternaryContext"&gt;The new rendering context.</param&gt;
00688     public void SetQuaternaryContext(Context quaternaryContext)
00689     {
00690         Context = quaternaryContext;
00691     }
00692
00693     /// &lt;summary&gt;
00694     /// Get the current rendering thread ID.
00695     /// &lt;/summary&gt;
00696     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00697     public int GetQuaternaryThreadId()
00698     {
00699         return ThreadId;
00700     }
00701
00702     /// &lt;summary&gt;
00703     /// Set the current rendering thread ID.
00704     /// &lt;/summary&gt;
00705     /// &lt;param name="quaternaryThreadId"&gt;The new rendering thread ID.</param&gt;
00706     public void SetQuaternaryThreadId(int quaternaryThreadId)
00707     {
00708         ThreadId = quaternaryThreadId;
00709     }
00710
00711     /// &lt;summary&gt;
00712     /// Get the current rendering context.
00713     /// &lt;/summary&gt;
00714     /// &lt;returns&gt;The current rendering context.</returns&gt;
00715     public Context GetQuinaryContext()
00716     {
00717         return Context;
00718     }
00719
00720     /// &lt;summary&gt;
00721     /// Set the current rendering context.
00722     /// &lt;/summary&gt;
00723     /// &lt;param name="quinaryContext"&gt;The new rendering context.</param&gt;
00724     public void SetQuinaryContext(Context quinaryContext)
00725     {
00726         Context = quinaryContext;
00727     }
00728
00729     /// &lt;summary&gt;
00730     /// Get the current rendering thread ID.
00731     /// &lt;/summary&gt;
00732     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00733     public int GetQuinaryThreadId()
00734     {
00735         return ThreadId;
00736     }
00737
00738     /// &lt;summary&gt;
00739     /// Set the current rendering thread ID.
00740     /// &lt;/summary&gt;
00741     /// &lt;param name="quinaryThreadId"&gt;The new rendering thread ID.</param&gt;
00742     public void SetQuinaryThreadId(int quinaryThreadId)
00743     {
00744         ThreadId = quinaryThreadId;
00745     }
00746
00747     /// &lt;summary&gt;
00748     /// Get the current rendering context.
00749     /// &lt;/summary&gt;
00750     /// &lt;returns&gt;The current rendering context.</returns&gt;
00751     public Context GetSenaryContext()
00752     {
00753         return Context;
00754     }
00755
00756     /// &lt;summary&gt;
00757     /// Set the current rendering context.
00758     /// &lt;/summary&gt;
00759     /// &lt;param name="senaryContext"&gt;The new rendering context.</param&gt;
00760     public void SetSenaryContext(Context senaryContext)
00761     {
00762         Context = senaryContext;
00763     }
00764
00765     /// &lt;summary&gt;
00766     /// Get the current rendering thread ID.
00767     /// &lt;/summary&gt;
00768     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00769     public int GetSenaryThreadId()
00770     {
00771         return ThreadId;
00772     }
00773
00774     /// &lt;summary&gt;
00775     /// Set the current rendering thread ID.
00776     /// &lt;/summary&gt;
00777     /// &lt;param name="senaryThreadId"&gt;The new rendering thread ID.</param&gt;
00778     public void SetSenaryThreadId(int senaryThreadId)
00779     {
00780         ThreadId = senaryThreadId;
00781     }
00782
00783     /// &lt;summary&gt;
00784     /// Get the current rendering context.
00785     /// &lt;/summary&gt;
00786     /// &lt;returns&gt;The current rendering context.</returns&gt;
00787     public Context GetSeptenaryContext()
00788     {
00789         return Context;
00790     }
00791
00792     /// &lt;summary&gt;
00793     /// Set the current rendering context.
00794     /// &lt;/summary&gt;
00795     /// &lt;param name="septenaryContext"&gt;The new rendering context.</param&gt;
00796     public void SetSeptenaryContext(Context septenaryContext)
00797     {
00798         Context = septenaryContext;
00799     }
00800
00801     /// &lt;summary&gt;
00802     /// Get the current rendering thread ID.
00803     /// &lt;/summary&gt;
00804     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00805     public int GetSeptenaryThreadId()
00806     {
00807         return ThreadId;
00808     }
00809
00810     /// &lt;summary&gt;
00811     /// Set the current rendering thread ID.
00812     /// &lt;/summary&gt;
00813     /// &lt;param name="septenaryThreadId"&gt;The new rendering thread ID.</param&gt;
00814     public void SetSeptenaryThreadId(int septenaryThreadId)
00815     {
00816         ThreadId = septenaryThreadId;
00817     }
00818
00819     /// &lt;summary&gt;
00820     /// Get the current rendering context.
00821     /// &lt;/summary&gt;
00822     /// &lt;returns&gt;The current rendering context.</returns&gt;
00823     public Context GetOctonaryContext()
00824     {
00825         return Context;
00826     }
00827
00828     /// &lt;summary&gt;
00829     /// Set the current rendering context.
00830     /// &lt;/summary&gt;
00831     /// &lt;param name="octonaryContext"&gt;The new rendering context.</param&gt;
00832     public void SetOctonaryContext(Context octonaryContext)
00833     {
00834         Context = octonaryContext;
00835     }
00836
00837     /// &lt;summary&gt;
00838     /// Get the current rendering thread ID.
00839     /// &lt;/summary&gt;
00840     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00841     public int GetOctonaryThreadId()
00842     {
00843         return ThreadId;
00844     }
00845
00846     /// &lt;summary&gt;
00847     /// Set the current rendering thread ID.
00848     /// &lt;/summary&gt;
00849     /// &lt;param name="octonaryThreadId"&gt;The new rendering thread ID.</param&gt;
00850     public void SetOctonaryThreadId(int octonaryThreadId)
00851     {
00852         ThreadId = octonaryThreadId;
00853     }
00854
00855     /// &lt;summary&gt;
00856     /// Get the current rendering context.
00857     /// &lt;/summary&gt;
00858     /// &lt;returns&gt;The current rendering context.</returns&gt;
00859     public Context GetNonaryContext()
00860     {
00861         return Context;
00862     }
00863
00864     /// &lt;summary&gt;
00865     /// Set the current rendering context.
00866     /// &lt;/summary&gt;
00867     /// &lt;param name="nonaryContext"&gt;The new rendering context.</param&gt;
00868     public void SetNonaryContext(Context nonaryContext)
00869     {
00870         Context = nonaryContext;
00871     }
00872
00873     /// &lt;summary&gt;
00874     /// Get the current rendering thread ID.
00875     /// &lt;/summary&gt;
00876     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00877     public int GetNonaryThreadId()
00878     {
00879         return ThreadId;
00880     }
00881
00882     /// &lt;summary&gt;
00883     /// Set the current rendering thread ID.
00884     /// &lt;/summary&gt;
00885     /// &lt;param name="nonaryThreadId"&gt;The new rendering thread ID.</param&gt;
00886     public void SetNonaryThreadId(int nonaryThreadId)
00887     {
00888         ThreadId = nonaryThreadId;
00889     }
00890
00891     /// &lt;summary&gt;
00892     /// Get the current rendering context.
00893     /// &lt;/summary&gt;
00894     /// &lt;returns&gt;The current rendering context.</returns&gt;
00895     public Context GetDecaryContext()
00896     {
00897         return Context;
00898     }
00899
00900     /// &lt;summary&gt;
00901     /// Set the current rendering context.
00902     /// &lt;/summary&gt;
00903     /// &lt;param name="decaryContext"&gt;The new rendering context.</param&gt;
00904     public void SetDecaryContext(Context decaryContext)
00905     {
00906         Context = decaryContext;
00907     }
00908
00909     /// &lt;summary&gt;
00910     /// Get the current rendering thread ID.
00911     /// &lt;/summary&gt;
00912     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00913     public int GetDecaryThreadId()
00914     {
00915         return ThreadId;
00916     }
00917
00918     /// &lt;summary&gt;
00919     /// Set the current rendering thread ID.
00920     /// &lt;/summary&gt;
00921     /// &lt;param name="decaryThreadId"&gt;The new rendering thread ID.</param&gt;
00922     public void SetDecaryThreadId(int decaryThreadId)
00923     {
00924         ThreadId = decaryThreadId;
00925     }
00926
00927     /// &lt;summary&gt;
00928     /// Get the current rendering context.
00929     /// &lt;/summary&gt;
00930     /// &lt;returns&gt;The current rendering context.</returns&gt;
00931     public Context GetUndecaryContext()
00932     {
00933         return Context;
00934     }
00935
00936     /// &lt;summary&gt;
00937     /// Set the current rendering context.
00938     /// &lt;/summary&gt;
00939     /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
00940     public void SetUndecaryContext(Context undecaryContext)
00941     {
00942         Context = undecaryContext;
00943     }
00944
00945     /// &lt;summary&gt;
00946     /// Get the current rendering thread ID.
00947     /// &lt;/summary&gt;
00948     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00949     public int GetUndecaryThreadId()
00950     {
00951         return ThreadId;
00952     }
00953
00954     /// &lt;summary&gt;
00955     /// Set the current rendering thread ID.
00956     /// &lt;/summary&gt;
00957     /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param&gt;
00958     public void SetUndecaryThreadId(int undecaryThreadId)
00959     {
00960         ThreadId = undecaryThreadId;
00961     }
00962
00963     /// &lt;summary&gt;
00964     /// Get the current rendering context.
00965     /// &lt;/summary&gt;
00966     /// &lt;returns&gt;The current rendering context.</returns&gt;
00967     public Context GetUndecaryContext()
00968     {
00969         return Context;
00970     }
00971
00972     /// &lt;summary&gt;
00973     /// Set the current rendering context.
00974     /// &lt;/summary&gt;
00975     /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
00976     public void SetUndecaryContext(Context undecaryContext)
00977     {
00978         Context = undecaryContext;
00979     }
00980
00981     /// &lt;summary&gt;
00982     /// Get the current rendering thread ID.
00983     /// &lt;/summary&gt;
00984     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
00985     public int GetUndecaryThreadId()
00986     {
00987         return ThreadId;
00988     }
00989
00990     /// &lt;summary&gt;
00991     /// Set the current rendering thread ID.
00992     /// &lt;/summary&gt;
00993     /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param&gt;
00994     public void SetUndecaryThreadId(int undecaryThreadId)
00995     {
00996         ThreadId = undecaryThreadId;
00997     }
00998
00999     /// &lt;summary&gt;
01000     /// Get the current rendering context.
01001     /// &lt;/summary&gt;
01002     /// &lt;returns&gt;The current rendering context.</returns&gt;
01003     public Context GetUndecaryContext()
01004     {
01005         return Context;
01006     }
01007
01008     /// &lt;summary&gt;
01009     /// Set the current rendering context.
01010     /// &lt;/summary&gt;
01011     /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
01012     public void SetUndecaryContext(Context undecaryContext)
01013     {
01014         Context = undecaryContext;
01015     }
01016
01017     /// &lt;summary&gt;
01018     /// Get the current rendering thread ID.
01019     /// &lt;/summary&gt;
01020     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
01021     public int GetUndecaryThreadId()
01022     {
01023         return ThreadId;
01024     }
01025
01026     /// &lt;summary&gt;
01027     /// Set the current rendering thread ID.
01028     /// &lt;/summary&gt;
01029     /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param&gt;
01030     public void SetUndecaryThreadId(int undecaryThreadId)
01031     {
01032         ThreadId = undecaryThreadId;
01033     }
01034
01035     /// &lt;summary&gt;
01036     /// Get the current rendering context.
01037     /// &lt;/summary&gt;
01038     /// &lt;returns&gt;The current rendering context.</returns&gt;
01039     public Context GetUndecaryContext()
01040     {
01041         return Context;
01042     }
01043
01044     /// &lt;summary&gt;
01045     /// Set the current rendering context.
01046     /// &lt;/summary&gt;
01047     /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
01048     public void SetUndecaryContext(Context undecaryContext)
01049     {
01050         Context = undecaryContext;
01051     }
01052
01053     /// &lt;summary&gt;
01054     /// Get the current rendering thread ID.
01055     /// &lt;/summary&gt;
01056     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
01057     public int GetUndecaryThreadId()
01058     {
01059         return ThreadId;
01060     }
01061
01062     /// &lt;summary&gt;
01063     /// Set the current rendering thread ID.
01064     /// &lt;/summary&gt;
01065     /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param&gt;
01066     public void SetUndecaryThreadId(int undecaryThreadId)
01067     {
01068         ThreadId = undecaryThreadId;
01069     }
01070
01071     /// &lt;summary&gt;
01072     /// Get the current rendering context.
01073     /// &lt;/summary&gt;
01074     /// &lt;returns&gt;The current rendering context.</returns&gt;
01075     public Context GetUndecaryContext()
01076     {
01077         return Context;
01078     }
01079
01080     /// &lt;summary&gt;
01081     /// Set the current rendering context.
01082     /// &lt;/summary&gt;
01083     /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
01084     public void SetUndecaryContext(Context undecaryContext)
01085     {
01086         Context = undecaryContext;
01087     }
01088
01089     /// &lt;summary&gt;
01090     /// Get the current rendering thread ID.
01091     /// &lt;/summary&gt;
01092     /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
01093     public int GetUndecaryThreadId()
01094     {
01095         return ThreadId;
01096     }
01097
01098     /// &lt;summary&gt;
01099     /// Set the current rendering thread ID.
01100    /// &lt;/summary&gt;
01101    /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param&gt;
01102    public void SetUndecaryThreadId(int undecaryThreadId)
01103    {
01104        ThreadId = undecaryThreadId;
01105    }
01106
01107    /// &lt;summary&gt;
01108    /// Get the current rendering context.
01109    /// &lt;/summary&gt;
01110    /// &lt;returns&gt;The current rendering context.</returns&gt;
01111    public Context GetUndecaryContext()
01112    {
01113        return Context;
01114    }
01115
01116    /// &lt;summary&gt;
01117    /// Set the current rendering context.
01118    /// &lt;/summary&gt;
01119    /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
01120    public void SetUndecaryContext(Context undecaryContext)
01121    {
01122        Context = undecaryContext;
01123    }
01124
01125    /// &lt;summary&gt;
01126    /// Get the current rendering thread ID.
01127    /// &lt;/summary&gt;
01128    /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
01129    public int GetUndecaryThreadId()
01130    {
01131        return ThreadId;
01132    }
01133
01134    /// &lt;summary&gt;
01135    /// Set the current rendering thread ID.
01136    /// &lt;/summary&gt;
01137    /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param&gt;
01138    public void SetUndecaryThreadId(int undecaryThreadId)
01139    {
01140        ThreadId = undecaryThreadId;
01141    }
01142
01143    /// &lt;summary&gt;
01144    /// Get the current rendering context.
01145    /// &lt;/summary&gt;
01146    /// &lt;returns&gt;The current rendering context.</returns&gt;
01147    public Context GetUndecaryContext()
01148    {
01149        return Context;
01150    }
01151
01152    /// &lt;summary&gt;
01153    /// Set the current rendering context.
01154    /// &lt;/summary&gt;
01155    /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
01156    public void SetUndecaryContext(Context undecaryContext)
01157    {
01158        Context = undecaryContext;
01159    }
01160
01161    /// &lt;summary&gt;
01162    /// Get the current rendering thread ID.
01163    /// &lt;/summary&gt;
01164    /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
01165    public int GetUndecaryThreadId()
01166    {
01167        return ThreadId;
01168    }
01169
01170    /// &lt;summary&gt;
01171    /// Set the current rendering thread ID.
01172    /// &lt;/summary&gt;
01173    /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param&gt;
01174    public void SetUndecaryThreadId(int undecaryThreadId)
01175    {
01176        ThreadId = undecaryThreadId;
01177    }
01178
01179    /// &lt;summary&gt;
01180    /// Get the current rendering context.
01181    /// &lt;/summary&gt;
01182    /// &lt;returns&gt;The current rendering context.</returns&gt;
01183    public Context GetUndecaryContext()
01184    {
01185        return Context;
01186    }
01187
01188    /// &lt;summary&gt;
01189    /// Set the current rendering context.
01190    /// &lt;/summary&gt;
01191    /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
01192    public void SetUndecaryContext(Context undecaryContext)
01193    {
01194        Context = undecaryContext;
01195    }
01196
01197    /// &lt;summary&gt;
01198    /// Get the current rendering thread ID.
01199    /// &lt;/summary&gt;
01200    /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
01201    public int GetUndecaryThreadId()
01202    {
01203        return ThreadId;
01204    }
01205
01206    /// &lt;summary&gt;
01207    /// Set the current rendering thread ID.
01208    /// &lt;/summary&gt;
01209    /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param&gt;
01210    public void SetUndecaryThreadId(int undecaryThreadId)
01211    {
01212        ThreadId = undecaryThreadId;
01213    }
01214
01215    /// &lt;summary&gt;
01216    /// Get the current rendering context.
01217    /// &lt;/summary&gt;
01218    /// &lt;returns&gt;The current rendering context.</returns&gt;
01219    public Context GetUndecaryContext()
01220    {
01221        return Context;
01222    }
01223
01224    /// &lt;summary&gt;
01225    /// Set the current rendering context.
01226    /// &lt;/summary&gt;
01227    /// &lt;param name="undecaryContext"&gt;The new rendering context.</param&gt;
01228    public void SetUndecaryContext(Context undecaryContext)
01229    {
01230        Context = undecaryContext;
01231    }
01232
01233    /// &lt;summary&gt;
01234    /// Get the current rendering thread ID.
01235    /// &lt;/summary&gt;
01236    /// &lt;returns&gt;The current rendering thread ID.</returns&gt;
01237    public int GetUndecaryThreadId()
01238    {
01239        return ThreadId;
01240    }
01241
01242    /// &lt;summary&gt;
01243    /// Set the current rendering thread ID.
01244    /// &lt;/summary&gt;
01245    /// &lt;param name="undecaryThreadId"&gt;The new rendering thread ID.</param>
01246    public void SetUndecaryThreadId(int undecaryThreadId)
01247    {
01248        ThreadId = undec
```

```

00443             }
00444         }
00445         else if (roundedOrigin.Height < targets[i].Height)
00446     {
00447         if (origins[i].X1 < actualPageArea.X1)
00448         {
00449             origins[i] = new Rectangle(origins[i].X0, origins[i].Y0, origins[i].X1,
00450                                         origins[i].Y1 + 0.5 / zoom);
00451         }
00452         else
00453         {
00454             origins[i] = new Rectangle(origins[i].X0, origins[i].Y0 - 0.5 / zoom,
00455                                         origins[i].X1, origins[i].Y1);
00456         }
00457         roundedOrigin = origins[i].Round(zoom);
00458         if (roundedOrigin.Width == oldRoundedOrigin.Width && roundedOrigin.Height ==
00459             oldRoundedOrigin.Height && (roundedOrigin.Width != targets[i].Width || roundedOrigin.Height !=
00460             targets[i].Height))
00461         {
00462             countBlanks++;
00463         }
00464         if (countBlanks >= 100)
00465         {
00466             //It seems that we can't coerce the expected size and the actual size to be
00467             the same. Give up.
00468             return;
00469         }
00470     }
00471
00472     //Start each rendering thread.
00473     for (int i = 0; i < destinations.Length; i++)
00474     {
00475         double dzoom = zoom;
00476         Rectangle origin = origins[i];
00477         if (this.ImageXRes != 72 || this.ImageYRes != 72)
00478         {
00479             dzoom *= Math.Sqrt(this.ImageXRes * this.ImageYRes) / 72;
00480             origin = new Rectangle(origin.X0 * 72 / this.ImageXRes, origin.Y0 * 72 /
00481             this.ImageYRes, origin.X1 * 72 / this.ImageXRes, origin.Y1 * 72 / this.ImageYRes);
00482         }
00483
00484         RenderingThreads[i].Render(Contexts[i].NativeContext, DisplayList, origin,
00485         (float)dzoom, destinations[i], pixelFormat, this.PageBounds, ClipToPageBounds);
00486     }
00487
00488     //Wait until all the rendering threads have finished.
00489     for (int i = 0; i < destinations.Length; i++)
00490     {
00491         RenderingThreads[i].WaitForRendering();
00492     }
00493
00494     /// <summary>
00495     /// Gets an element from a collection of <see cref="System.Span{T}">Span</see>&lt;see
00496     /// cref="byte">/&gt;;
00497     /// </summary>
00498     /// <param name="index">The index of the element to get.</param>
00499     /// <returns>An element from a collection of <see cref="System.Span{T}">Span</see>&lt;see
00500     /// cref="byte">/&gt;;</returns>
00501     public delegate Span<byte> GetSpanItem(int index);

00502     /// <summary>
00503     /// Render the specified region to an image of the specified size, split in a number of tiles
00504     /// equal to the number of threads used by this <see cref="MuPDFMultiThreadedPageRenderer"/>, without
00505     /// marshaling. This method will not return until all the rendering threads have finished.
00506     /// Since creating an array of <see cref="Span{T}"> is not allowed, this method returns a
00507     /// delegate that accepts an integer parameter (representing the index of the span in the "array") and
00508     /// returns the <see cref="Span{T}"> corresponding to that index.
00509     /// </summary>
00510     /// <param name="targetSize">The total size of the image that should be rendered.</param>
00511     /// <param name="region">The region in page units that should be rendered.</param>
00512     /// <param name="disposables">A collection of <see cref="IDisposable"/>s that can be used to
00513     /// free the memory where the rendered tiles are stored. You should keep track of these and dispose them
00514     /// when you have finished working with the images.</param>
00515     /// <param name="pixelFormat">The format of the pixel data.</param>
00516     /// <returns>A delegate that accepts an integer parameter (representing the index of the span
00517     /// in the "array") and returns the <see cref="Span{T}"> corresponding to that index.</returns>
00518     public GetSpanItem Render(RoundedSize targetSize, Rectangle region, out IDisposable[]
00519     disposables, PixelFormats pixelFormat)
00520     {
00521         RoundedRectangle[] targets = targetSize.Split(this.ThreadCount);
00522     }

```

```

00513     IntPtr[] destinations = new IntPtr[targets.Length];
00514     disposables = new IDisposable[targets.Length];
00515 
00516     bool hasAlpha = pixelFormat == PixelFormats.RGBA || pixelFormat == PixelFormats.BGRA;
00517 
00518     int[] sizes = new int[destinations.Length];
00519 
00520     for (int i = 0; i < targets.Length; i++)
00521     {
00522         int allocSize = targets[i].Width * targets[i].Height * (hasAlpha ? 4 : 3);
00523 
00524         sizes[i] = allocSize;
00525         destinations[i] = Marshal.AllocHGlobal(allocSize);
00526         disposables[i] = new DisposableIntPtr(destinations[i], allocSize);
00527     }
00528 
00529     this.Render(targetSize, region, destinations, pixelFormat);
00530 
00531     return i =>
00532     {
00533         unsafe
00534         {
00535             return new Span<byte>((void*)destinations[i], sizes[i]);
00536         }
00537     };
00538 }
00539 
00540 /// <summary>
00541 /// Signal to the rendering threads that they should abort rendering as soon as possible.
00542 /// </summary>
00543 public void Abort()
00544 {
00545     for (int i = 0; i < RenderingThreads.Length; i++)
00546     {
00547         RenderingThreads[i].AbortRendering();
00548     }
00549 }
00550 
00551 /// <summary>
00552 /// Get the current rendering progress of all the threads.
00553 /// </summary>
00554 /// <returns>A <see cref="RenderProgress"/> object containing the rendering progress of all
00555 // the threads.</returns>
00556 public RenderProgress GetProgress()
00557 {
00558     return new RenderProgress((from el in RenderingThreads select
00559     el.GetProgress()).ToArray());
00560 }
00561 
00562 private bool disposedValue;
00563 
00564 protected virtual void Dispose(bool disposing)
00565 {
00566     if (!disposedValue)
00567     {
00568         if (disposing)
00569         {
00570             Abort();
00571 
00572             if (RenderingThreads != null)
00573             {
00574                 for (int i = 0; i < RenderingThreads.Length; i++)
00575                 {
00576                     RenderingThreads[i].Dispose();
00577                 }
00578 
00579             if (Contexts != null)
00580             {
00581                 for (int i = 0; i < Contexts.Length; i++)
00582                 {
00583                     Contexts[i].Dispose();
00584                 }
00585             }
00586         }
00587         disposedValue = true;
00588     }
00589 }
00590 
00591 public void Dispose()
00592 {
00593     Dispose(disposing: true);
00594     GC.SuppressFinalize(this);
00595 }
00596 }
00597 }
```

```
00598 }
```

8.9 MuPDFPage.cs

```
00001 /*
00002     MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
00003     Copyright (C) 2020 Giorgio Bianchini
00004
00005     This program is free software: you can redistribute it and/or modify
00006     it under the terms of the GNU Affero General Public License as
00007     published by the Free Software Foundation, version 3.
00008
00009     This program is distributed in the hope that it will be useful,
00010     but WITHOUT ANY WARRANTY; without even the implied warranty of
00011     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012     GNU Affero General Public License for more details.
00013
00014     You should have received a copy of the GNU Affero General Public License
00015     along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using System;
00019 using System.Collections;
00020 using System.Collections.Generic;
00021
00022 namespace MuPDFCore
00023 {
00024     /// <summary>
00025     /// A wrapper over a MuPDF page object, which contains information about the page's boundaries.
00026     /// </summary>
00027     public class MuPDFPage : IDisposable
00028     {
00029         /// <summary>
00030         /// The page's bounds at 72 DPI. Read-only.
00031         /// </summary>
00032         public Rectangle Bounds { get; }
00033
00034         /// <summary>
00035         /// The number of this page in the original document.
00036         /// </summary>
00037         public int PageNumber { get; }
00038
00039         /// <summary>
00040         /// A pointer to the native page object.
00041         /// </summary>
00042         internal readonly IntPtr NativePage;
00043
00044         /// <summary>
00045         /// The context that owns the document from which this page was extracted.
00046         /// </summary>
00047         private readonly MuPDFContext OwnerContext;
00048
00049         /// <summary>
00050         /// The document from which the page was extracted.
00051         /// </summary>
00052         internal readonly MuPDFDocument OwnerDocument;
00053
00054         /// <summary>
00055         /// The page's original bounds. Read-only.
00056         /// </summary>
00057         internal Rectangle OriginalBounds { get; }
00058
00059         /// <summary>
00060         /// Create a new <see cref="MuPDFPage"/> object from the specified document.
00061         /// </summary>
00062         /// <param name="context">The context that owns the document.</param>
00063         /// <param name="document">The document from which the page should be extracted.</param>
00064         /// <param name="number">The number of the page that should be extracted (starting at
00065         0).</param>
00066         internal MuPDFPage(MuPDFContext context, MuPDFDocument document, int number)
00067         {
00068             if (document.EncryptionState == EncryptionState.Encrypted)
00069             {
00070                 throw new DocumentLockedException("A password is necessary to render the document!");
00071             }
00072
00073             this.OwnerContext = context;
00074             this.OwnerDocument = document;
00075             this.PageNumber = number;
00076
00077             float x = 0;
00078             float y = 0;
00079             float w = 0;
```

```

00079         float h = 0;
00080
00081     ExitCodes result = (ExitCodes)NativeMethods.LoadPage(context.NativeContext,
00082     document.NativeDocument, number, ref NativePage, ref x, ref y, ref w, ref h);
00083
00084     this.Bounds = new Rectangle(Math.Round(x * document.ImageXRes / 72.0 * 1000) / 1000,
00085     Math.Round(y * document.ImageYRes / 72.0 * 1000) / 1000, Math.Round(w * document.ImageXRes / 72.0 *
00086     1000) / 1000, Math.Round(h * document.ImageYRes / 72.0 * 1000) / 1000);
00087     this.OriginalBounds = new Rectangle(x, y, w, h);
00088
00089     switch (result)
00090     {
00091         case ExitCodes.EXIT_SUCCESS:
00092             break;
00093         case ExitCodes.ERR_CANNOT_LOAD_PAGE:
00094             throw new MuPDFException("Cannot load page", result);
00095         case ExitCodes.ERR_CANNOT_COMPUTE_BOUNDS:
00096             throw new MuPDFException("Cannot compute bounds", result);
00097         default:
00098             throw new MuPDFException("Unknown error", result);
00099     }
00100 }
00101
00102 private bool disposedValue;
00103
00104 //;<inheritdoc>
00105 protected virtual void Dispose(bool disposing)
00106 {
00107     if (!disposedValue)
00108     {
00109         NativeMethods.DisposePage(OwnerContext.NativeContext, NativePage);
00110         disposedValue = true;
00111     }
00112 }
00113 ~MuPDFPage()
00114 {
00115     if (NativePage != IntPtr.Zero)
00116     {
00117         Dispose(disposing: false);
00118     }
00119 }
00120
00121 //;<inheritdoc>
00122 public void Dispose()
00123 {
00124     Dispose(disposing: true);
00125     GC.SuppressFinalize(this);
00126 }
00127
00128 /// <summary>
00129 /// A lazy collection of <see cref="MuPDFPage"/>s. Each page is loaded from the document as it is
00130 requested for the first time.
00131 /// </summary>
00132 public class MuPDFPageCollection : IReadOnlyList<MuPDFPage>, IDisposable
00133 {
00134     /// <summary>
00135     /// The internal store of the pages.
00136     private readonly MuPDFPage[] Pages;
00137
00138     /// <summary>
00139     /// The context that owns the document from which the pages were extracted.
00140     /// </summary>
00141     private readonly MuPDFContext OwnerContext;
00142
00143     /// <summary>
00144     /// The document from which the pages were extracted.
00145     /// </summary>
00146     private readonly MuPDFDocument OwnerDocument;
00147
00148     /// <summary>
00149     /// The number of pages in the collection.
00150     /// </summary>
00151     public int Length { get { return Pages.Length; } }
00152
00153     /// <summary>
00154     /// The number of pages in the collection.
00155     /// </summary>
00156     public int Count { get { return Pages.Length; } }
00157
00158     /// <summary>
00159     /// Get a page from the collection.
00160     /// </summary>
00161     /// <param name="index">The number of the page (starting at 0).</param>

```

```

00162     /// <returns>The specified <see cref="MuPDFPage"/>. </returns>
00163     public MuPDFPage this[int index]
00164     {
00165         get
00166         {
00167             if (index < 0 || index > Pages.Length)
00168             {
00169                 throw new IndexOutOfRangeException();
00170             }
00171
00172             if (Pages[index] == null)
00173             {
00174                 Pages[index] = new MuPDFPage(OwnerContext, OwnerDocument, index);
00175             }
00176
00177             return Pages[index];
00178         }
00179     }
00180
00181     /// <summary>
00182     /// Create a new <see cref="MuPDFPageCollection"/> from the specified document, containing the
00183     /// specified number of pages.
00184     /// </summary>
00185     /// <param name="context">The context that owns the document.</param>
00186     /// <param name="document">The document from which the pages should be extracted.</param>
00187     /// <param name="length">The number of pages in the document.</param>
00188     internal MuPDFPageCollection(MuPDFContext context, MuPDFDocument document, int length)
00189     {
00190         Pages = new MuPDFPage[length];
00191         OwnerContext = context;
00192         OwnerDocument = document;
00193     }
00194
00195     /// <inheritDoc/>
00196     public IEnumarator<MuPDFPage> GetEnumerator()
00197     {
00198         for (int i = 0; i < Pages.Length; i++)
00199         {
00200             if (Pages[i] == null)
00201             {
00202                 Pages[i] = new MuPDFPage(OwnerContext, OwnerDocument, i);
00203             }
00204
00205             return ((IEnumarable<MuPDFPage>)Pages).GetEnumerator();
00206         }
00207
00208         /// <inheritDoc/>
00209         IEnumarable.GetEnumerator()
00210         {
00211             return GetEnumerator();
00212         }
00213
00214         private bool disposedValue;
00215
00216         /// <inheritDoc/>
00217         protected virtual void Dispose(bool disposing)
00218         {
00219             if (!disposedValue)
00220             {
00221                 if (disposing)
00222                 {
00223                     for (int i = 0; i < Pages.Length; i++)
00224                     {
00225                         Pages[i]?.Dispose();
00226                     }
00227
00228                     disposedValue = true;
00229                 }
00230             }
00231         }
00232
00233         /// <inheritDoc/>
00234         public void Dispose()
00235         {
00236             Dispose(disposing: true);
00237             GC.SuppressFinalize(this);
00238         }
00239     }
00240 }

```

8.10 MuPDFStructuredTextPage.cs

```
00001 using System;
```

```

00002 using System.Collections;
00003 using System.Collections.Generic;
00004 using System.Runtime.InteropServices;
00005 using System.Text;
00006 using System.Text.RegularExpressions;
00007 using System.Threading;
00008 using System.Threading.Tasks;
00009
00010 namespace MuPDFCore
00011 {
00012     /// <summary>
00013     /// Describes OCR progress.
00014     /// </summary>
00015     public class OCRProgressInfo
00016     {
00017         /// <summary>
00018         /// A value between 0 and 1, indicating how much progress has been completed.
00019         /// </summary>
00020         public double Progress { get; }
00021
00022         internal OCRProgressInfo(double progress)
00023         {
00024             this.Progress = progress;
00025         }
00026     }
00027
00028     /// <summary>
00029     /// Represents a structured representation of the text contained in a page.
00030     /// </summary>
00031     public class MuPDFStructuredTextPage : IReadOnlyList<MuPDFStructuredTextBlock>
00032     {
00033         /// <summary>
00034         /// The blocks contained in the page.
00035         /// </summary>
00036         public MuPDFStructuredTextBlock[] StructuredTextBlocks { get; private set; }
00037
00038         /// <summary>
00039         /// The number of blocks in the page.
00040         /// </summary>
00041         public int Count =>
00042             ((IReadOnlyCollection<MuPDFStructuredTextBlock>)StructuredTextBlocks).Count;
00043
00044         /// <summary>
00045         /// Gets the specified block in the page.
00046         /// </summary>
00047         /// <param name="index">The index of the block.</param>
00048         /// <returns>The block with the specified <paramref name="index"/>.</returns>
00049         public MuPDFStructuredTextBlock this[int index] =>
00050             ((IReadOnlyList<MuPDFStructuredTextBlock>)StructuredTextBlocks)[index];
00051
00052         /// <summary>
00053         /// Gets the specified character in the page.
00054         /// </summary>
00055         /// <param name="address">The address (block, line and character index) of the
00056         /// character.</param>
00057         /// <returns>A <see cref="MuPDFStructuredTextCharacter"/> representing the specified
00058         /// character.</returns>
00059         public MuPDFStructuredTextCharacter this[MuPDFStructuredTextAddress address]
00060         {
00061             get
00062             {
00063                 return
00064                     StructuredTextBlocks[address.BlockIndex][address.LineIndex][address.CharacterIndex];
00065             }
00066         }
00067
00068         internal MuPDFStructuredTextPage(MuPDFContext context, MuPDFDisplayList list,
00069             TesseractLanguage ocrLanguage, double zoom, Rectangle pageBounds, CancellationToken cancellationToken
00070             = default, IProgress<OCRProgressInfo> progress = null)
00071         {
00072             if (ocrLanguage != null && RuntimeInformation.IsOSPlatform(OSPlatform.Windows) &&
00073                 RuntimeInformation.ProcessArchitecture == Architecture.X86 && (cancellationToken != default ||
00074                 progress != null))
00075             {
00076                 throw new PlatformNotSupportedException("A cancellationToken or a progress callback
00077                 are not supported on Windows x86!");
00078             }
00079
00080             int blockCount = -1;
00081
00082             IntPtr nativeStructuredPage = IntPtr.Zero;
00083
00084             ExitCodes result;
00085
00086             if (ocrLanguage != null)
00087             {
00088                 result = (ExitCodes)NativeMethods.GetStructuredTextPageWithOCR(context.NativeContext,

```

```

list.NativeDisplayList, ref nativeStructuredPage, ref blockCount, (float)zoom, pageBounds.X0,
pageBounds.Y0, pageBounds.X1, pageBounds.Y1, "TESSDATA_PREFIX=" + ocrLanguage.Prefix,
ocrLanguage.Language, prog =>
00079     {
00080         progress?.Report(new OCRProgressInfo(prog / 100.0));
00081
00082         if (cancellationToken.IsCancellationRequested)
00083         {
00084             return 1;
00085         }
00086         else
00087         {
00088             return 0;
00089         }
00090     });
00091 }
00092 else
00093 {
00094     result = (ExitCodes)NativeMethods.GetStructuredTextPage(context.NativeContext,
list.NativeDisplayList, ref nativeStructuredPage, ref blockCount);
00095 }
00096
00097 cancellationToken.ThrowIfCancellationRequested();
00098
00099 switch (result)
00100 {
00101     case ExitCodes.EXIT_SUCCESS:
00102         break;
00103     case ExitCodes.ERR_CANNOT_CREATE_PAGE:
00104         throw new MuPDFException("Cannot create page", result);
00105     case ExitCodes.ERR_CANNOT_POPULATE_PAGE:
00106         throw new MuPDFException("Cannot populate page", result);
00107     default:
00108         throw new MuPDFException("Unknown error", result);
00109 }
00110
00111 IntPtr[] blockPointers = new IntPtr[blockCount];
00112 GCHandle blocksHandle = GCHandle.Alloc(blockPointers, GCHandleType.Pinned);
00113
00114 try
00115 {
00116     result = (ExitCodes)NativeMethods.GetStructuredTextBlocks(nativeStructuredPage,
blocksHandle.AddrOfPinnedObject());
00117
00118     switch (result)
00119     {
00120         case ExitCodes.EXIT_SUCCESS:
00121             break;
00122         default:
00123             throw new MuPDFException("Unknown error", result);
00124     }
00125
00126     StructuredTextBlocks = new MuPDFStructuredTextBlock[blockCount];
00127
00128     for (int i = 0; i < blockCount; i++)
00129     {
00130         int type = -1;
00131         float x0 = -1;
00132         float y0 = -1;
00133         float x1 = -1;
00134         float y1 = -1;
00135         int lineCount = -1;
00136
00137         result = (ExitCodes)NativeMethods.GetStructuredTextBlock(blockPointers[i], ref
type, ref x0, ref y0, ref x1, ref y1, ref lineCount);
00138
00139         switch (result)
00140         {
00141             case ExitCodes.EXIT_SUCCESS:
00142                 break;
00143             default:
00144                 throw new MuPDFException("Unknown error", result);
00145         }
00146
00147         Rectangle bBox = new Rectangle(x0, y0, x1, y1);
00148
00149         switch ((MuPDFStructuredTextBlock.Types)type)
00150         {
00151             case MuPDFStructuredTextBlock.Types.Image:
00152                 this.StructuredTextBlocks[i] = new MuPDFImageStructuredTextBlock(bBox);
00153                 break;
00154             case MuPDFStructuredTextBlock.Types.Text:
00155                 this.StructuredTextBlocks[i] = new MuPDFTextStructuredTextBlock(bBox,
blockPointers[i], lineCount);
00156                 break;
00157         }
00158     }
00159 }

```

```

00159         }
00160     }
00161     {
00162         blocksHandle.Free();
00163     }
00164     NativeMethods.DisposeStructuredTextPage(context.NativeContext, nativeStructuredPage);
00165 }
00166 }
00167
00168 /// <summary>
00169 /// Gets the address of the character that contains the specified <paramref name="point"/> in
00170 page units.
00171     /// </summary>
00172     /// <param name="point">The point that must be contained by the character. This is expressed
00173 in page units (i.e. with a zoom factor of 1).</param>
00174     /// <param name="includeImages">If this is <see langword="true"/>, blocks containing images
00175 may be returned. Otherwise, only blocks containing text are considered.</param>
00176     /// <returns>The address of the character containing the specified <paramref name="point"/>,
00177 or <see langword="null"/> if no character contains the <paramref name="point"/>.</returns>
00178 public MuPDFStructuredTextAddress? GetHitAddress(PointF point, bool includeImages)
00179 {
00180     for (int i = 0; i < this.Count; i++)
00181     {
00182         if (includeImages || this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00183         {
00184             if (this[i].BoundingBox.Contains(point))
00185             {
00186                 for (int j = 0; j < this[i].Count; j++)
00187                 {
00188                     if (this[i][j].BoundingBox.Contains(point))
00189                     {
00190                         for (int k = 0; k < this[i][j].Count; k++)
00191                         {
00192                             if (this[i][j][k].BoundingQuad.Contains(point))
00193                             {
00194                                 return new MuPDFStructuredTextAddress(i, j, k);
00195                             }
00196                         }
00197                     }
00198                 }
00199             }
00200         }
00201
00202     /// <summary>
00203     /// Gets the address of the character that contains the specified <paramref name="point"/> in
00204 page units.
00205     /// </summary>
00206     /// <param name="point">The point that must be closest to the character. This is expressed in
00207 page units (i.e. with a zoom factor of 1).</param>
00208     /// <param name="includeImages">If this is <see langword="true"/>, blocks containing images
00209 may be returned. Otherwise, only blocks containing text are considered.</param>
00210     /// <returns>The address of the character closest to the specified <paramref name="point"/>
00211 This is <see langword="null"/> only if the page contains no characters.</returns>
00212 public MuPDFStructuredTextAddress? GetClosestHitAddress(PointF point, bool includeImages)
00213 {
00214     float minDistance = float.MaxValue;
00215     MuPDFStructuredTextAddress? closestHit = null;
00216
00217     float minBlockDistance = float.MaxValue;
00218     float minLineDistance = float.MaxValue;
00219
00220     for (int i = 0; i < this.Count; i++)
00221     {
00222         if (includeImages || this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00223         {
00224             float dx = Math.Max(0, Math.Max(this[i].BoundingBox.X0 - point.X, point.X -
00225 this[i].BoundingBox.X1));
00226             float dy = Math.Max(0, Math.Max(this[i].BoundingBox.Y0 - point.Y, point.Y -
00227 this[i].BoundingBox.Y1));
00228             float blockDist = dx * dx + dy * dy;
00229
00230             if (this[i].BoundingBox.Contains(point) || blockDist < minBlockDistance)
00231             {
00232                 if (blockDist < minBlockDistance)
00233                 {
00234                     minBlockDistance = blockDist;
00235                     minLineDistance = float.MaxValue;
00236                 }
00237
00238                 for (int j = 0; j < this[i].Count; j++)
00239                 {
00240                     dx = Math.Max(0, Math.Max(this[i][j].BoundingBox.X0 - point.X, point.X -
00241 this[i][j].BoundingBox.X1));
00242
00243                     if (this[i][j].BoundingBox.Contains(point) || dx < minLineDistance)
00244                     {
00245                         minLineDistance = dx;
00246                     }
00247                 }
00248             }
00249         }
00250     }
00251
00252     return closestHit;
00253 }

```

```

00235     dy = Math.Max(0, Math.Max(this[i][j].BoundingBox.Y0 - point.Y, point.Y -
00236         this[i][j].BoundingBox.Y1));
00237         float lineDist = dx * dx + dy * dy;
00238         if (this[i][j].BoundingBox.Contains(point) || lineDist < minLineDistance)
00239         {
00240             if (lineDist < minLineDistance)
00241             {
00242                 minLineDistance = lineDist;
00243             }
00244
00245             for (int k = 0; k < this[i][j].Count; k++)
00246             {
00247                 if (this[i][j][k].BoundingQuad.Contains(point))
00248                 {
00249                     return new MuPDFStructuredTextAddress(i, j, k);
00250                 }
00251                 else
00252                 {
00253                     //The quads should be small enough that the error due to only
00254                     //checking vertices and not sides is negligible. Also, since the square root is monotonous, we can skip
00255                     //it.
00256                     float minDist = (point.X -
00257                         this[i][j][k].BoundingQuad.UpperLeft.X) * (point.X - this[i][j][k].BoundingQuad.UpperLeft.X) +
00258                         (point.Y - this[i][j][k].BoundingQuad.UpperLeft.Y) * (point.Y -
00259                         this[i][j][k].BoundingQuad.UpperLeft.Y);
00260                     minDist = Math.Min(minDist, (point.X -
00261                         this[i][j][k].BoundingQuad.UpperRight.X) * (point.X - this[i][j][k].BoundingQuad.UpperRight.X) +
00262                         (point.Y - this[i][j][k].BoundingQuad.UpperRight.Y) * (point.Y -
00263                         this[i][j][k].BoundingQuad.UpperRight.Y));
00264                     minDist = Math.Min(minDist, (point.X -
00265                         this[i][j][k].BoundingQuad.LowerRight.X) * (point.X - this[i][j][k].BoundingQuad.LowerRight.X) +
00266                         (point.Y - this[i][j][k].BoundingQuad.LowerRight.Y) * (point.Y -
00267                         this[i][j][k].BoundingQuad.LowerRight.Y));
00268                     minDist = Math.Min(minDist, (point.X -
00269                         this[i][j][k].BoundingQuad.LowerLeft.X) * (point.X - this[i][j][k].BoundingQuad.LowerLeft.X) +
00270                         (point.Y - this[i][j][k].BoundingQuad.LowerLeft.Y) * (point.Y -
00271                         this[i][j][k].BoundingQuad.LowerLeft.Y));
00272
00273                     if (minDist < minDistance)
00274                     {
00275                         minDistance = minDist;
00276                         closestHit = new MuPDFStructuredTextAddress(i, j, k);
00277                     }
00278                 }
00279             }
00280         }
00281         return closestHit;
00282     }
00283
00284     /// <summary>
00285     /// Gets a collection of <see cref="Quad"/>s delimiting the specified character <paramref
00286     /// name="range"/>. Where possible, these are collapsed at the line and block level. Each <see
00287     /// cref="Quad"/> may or may not be a rectangle.
00288     /// </summary>
00289     /// <param name="range">A <see cref="MuPDFStructuredTextAddressSpan"/> representing the
00290     /// character range</param>
00291     /// <param name="includeImages">If this is <see langword="true"/>, the bounding boxes for
00292     /// blocks containing images are also returned. Otherwise, only blocks containing text are
00293     /// considered.</param>
00294     /// <returns>A lazy collection of <see cref="Quad"/>s delimiting the characters in the
00295     /// specified <paramref name="includeImages"/>.</returns>
00296     public IEnumerable<Quad> GetHighlightQuads(MuPDFStructuredTextAddressSpan range, bool
00297     includeImages)
00298     {
00299         if (range == null || range.End == null)
00300         {
00301             yield break;
00302         }
00303
00304         MuPDFStructuredTextAddress rangeStart = range.Start;
00305         MuPDFStructuredTextAddress rangeEnd = range.End.Value;
00306
00307         if (rangeEnd < rangeStart)
00308         {
00309             MuPDFStructuredTextAddress temp = rangeStart;
00310             rangeStart = rangeEnd;
00311             rangeEnd = temp;
00312         }
00313
00314         if (rangeStart.BlockIndex != rangeEnd.BlockIndex)
00315         {

```

```
00300         //Add remaining part of this block
00301         if (rangeStart.LineIndex == 0 && rangeStart.CharacterIndex == 0)
00302         {
00303             if (includeImages || this[rangeStart.BlockIndex].Type ==
MuPDFStructuredTextBlock.Types.Text)
00304             {
00305                 yield return this[rangeStart.BlockIndex].BoundingBox.ToQuad();
00306             }
00307         }
00308     }
00309     {
00310         if (rangeStart.CharacterIndex == 0)
00311         {
00312             yield return
this[rangeStart.BlockIndex][rangeStart.LineIndex].BoundingBox.ToQuad();
00313         }
00314         else
00315         {
00316             for (int i = rangeStart.CharacterIndex; i <
this[rangeStart.BlockIndex][rangeStart.LineIndex].Count; i++)
00317             {
00318                 yield return
this[rangeStart.BlockIndex][rangeStart.LineIndex][i].BoundingQuad;
00319             }
00320         }
00321
00322         for (int j = rangeStart.LineIndex + 1; j < this[rangeStart.BlockIndex].Count; j++)
00323         {
00324             yield return this[rangeStart.BlockIndex][j].BoundingBox.ToQuad();
00325         }
00326     }
00327
00328     //Add full blocks in the middle
00329     for (int i = rangeStart.BlockIndex + 1; i < rangeEnd.BlockIndex; i++)
00330     {
00331         if (includeImages || this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00332         {
00333             yield return this[i].BoundingBox.ToQuad();
00334         }
00335     }
00336
00337     rangeStart = new MuPDFStructuredTextAddress(rangeEnd.BlockIndex, 0, 0);
00338 }
00339
00340     if (includeImages || this[rangeStart.BlockIndex].Type ==
MuPDFStructuredTextBlock.Types.Text)
00341     {
00342         if (rangeStart.LineIndex != rangeEnd.LineIndex)
00343         {
00344             //Add remaining part of this line
00345             if (rangeStart.CharacterIndex == 0)
00346             {
00347                 yield return
this[rangeStart.BlockIndex][rangeStart.LineIndex].BoundingBox.ToQuad();
00348             }
00349             else
00350             {
00351                 for (int i = rangeStart.CharacterIndex; i <
this[rangeStart.BlockIndex][rangeStart.LineIndex].Count; i++)
00352                 {
00353                     yield return
this[rangeStart.BlockIndex][rangeStart.LineIndex][i].BoundingQuad;
00354                 }
00355             }
00356
00357             //Add full lines in the middle
00358             for (int j = rangeStart.LineIndex + 1; j < rangeEnd.LineIndex; j++)
00359             {
00360                 yield return this[rangeStart.BlockIndex][j].BoundingBox.ToQuad();
00361             }
00362
00363             rangeStart = new MuPDFStructuredTextAddress(rangeEnd.BlockIndex,
rangeEnd.LineIndex, 0);
00364         }
00365
00366         //Add remaining part of this line
00367         if (rangeStart.CharacterIndex == 0 && rangeEnd.CharacterIndex ==
this[rangeStart.BlockIndex][rangeStart.LineIndex].Count - 1)
00368         {
00369             yield return
this[rangeStart.BlockIndex][rangeStart.LineIndex].BoundingBox.ToQuad();
00370         }
00371         else
00372         {
00373             for (int j = rangeStart.CharacterIndex; j <= rangeEnd.CharacterIndex; j++)
00374             {
00375                 yield return
```

```

        this[rangeStart.BlockIndex][rangeStart.LineIndex][j].BoundingQuad;
00376            }
00377        }
00378    }
00379}
00380
00381    /// <summary>
00382    /// Gets the text corresponding to the specified character <paramref name="range"/>. Blocks
00383    /// containing images are ignored.
00384    /// </summary>
00385    /// <param name="range">A <see cref="MuPDFStructuredTextAddressSpan"/> representing the range
00386    /// of text to extract.</param>
00387    /// <returns>A string representation of the text contained in the specified <paramref
00388    /// name="range"/>.</returns>
00389    public string GetText(MuPDFStructuredTextAddressSpan range)
00390    {
00391        if (range == null || range.End == null)
00392        {
00393            return null;
00394        }
00395
00396        MuPDFStructuredTextAddress selectionStart = range.Start;
00397        MuPDFStructuredTextAddress selectionEnd = range.End.Value;
00398
00399        if (selectionEnd < selectionStart)
00400        {
00401            MuPDFStructuredTextAddress temp = selectionStart;
00402            selectionStart = selectionEnd;
00403            selectionEnd = temp;
00404        }
00405
00406        StringBuilder builder = new StringBuilder();
00407
00408        if (selectionStart.BlockIndex != selectionEnd.BlockIndex)
00409        {
00410            //Add remaining part of this block
00411            if (selectionStart.LineIndex == 0 && selectionStart.CharacterIndex == 0)
00412            {
00413                builder.Append(((MuPDFTextStructuredTextBlock)this[selectionStart.BlockIndex]).ToString());
00414            }
00415            else
00416            {
00417                if (selectionStart.CharacterIndex == 0)
00418                {
00419
00420                    builder.AppendLine(this[selectionStart.BlockIndex][selectionStart.LineIndex].ToString());
00421
00422                    for (int i = selectionStart.CharacterIndex; i <
00423 this[selectionStart.BlockIndex][selectionStart.LineIndex].Count; i++)
00424                    {
00425
00426                        builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex][i].ToString());
00427
00428                    }
00429
00430                    for (int j = selectionStart.LineIndex + 1; j <
00431 this[selectionStart.BlockIndex].Count; j++)
00432                    {
00433                        builder.AppendLine(this[selectionStart.BlockIndex][j].ToString());
00434                    }
00435
00436                    //Add full blocks in the middle
00437                    for (int i = selectionStart.BlockIndex + 1; i < selectionEnd.BlockIndex; i++)
00438                    {
00439                        if (this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00440                        {
00441                            builder.Append(this[i].ToString());
00442                        }
00443
00444
00445                        selectionStart = new MuPDFStructuredTextAddress(selectionEnd.BlockIndex, 0, 0);
00446                    }
00447
00448                    if (this[selectionStart.BlockIndex].Type == MuPDFStructuredTextBlock.Types.Text)
00449                    {
00450                        if (selectionStart.LineIndex != selectionEnd.LineIndex)
00451                        {
00452                            //Add remaining part of this line
00453                            if (selectionStart.CharacterIndex == 0)
00454
00455                                builder.AppendLine(this[selectionStart.BlockIndex][selectionStart.LineIndex].ToString());
00456
00457                            for (int i = selectionStart.CharacterIndex; i <
00458 this[selectionStart.BlockIndex][selectionStart.LineIndex].Count; i++)
00459                            {
00460
00461                                builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex][i].ToString());
00462
00463                            }
00464
00465                            selectionStart = new MuPDFStructuredTextAddress(selectionEnd.BlockIndex, 0, 0);
00466                        }
00467
00468                    }
00469
00470                }
00471            }
00472        }
00473    }
00474}

```

```

00454             {
00455                 builder.AppendLine(this[selectionStart.BlockIndex][selectionStart.LineIndex].ToString());
00456             }
00457             else
00458             {
00459                 for (int i = selectionStart.CharacterIndex; i <
this[selectionStart.BlockIndex][selectionStart.LineIndex].Count; i++)
00460                 {
00461                     builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex][i].ToString());
00462                 }
00463                     builder.AppendLine();
00464                 }
00465             }
00466             //Add full lines in the middle
00467             for (int j = selectionStart.LineIndex + 1; j < selectionEnd.LineIndex; j++)
00468             {
00469                 builder.AppendLine(this[selectionStart.BlockIndex][j].ToString());
00470             }
00471             selectionStart = new MuPDFStructuredTextAddress(selectionEnd.BlockIndex,
selectionEnd.LineIndex, 0);
00472         }
00473     }
00474     //Add remaining part of this line
00475     if (selectionStart.CharacterIndex == 0 && selectionEnd.CharacterIndex ==
this[selectionStart.BlockIndex][selectionStart.LineIndex].Count - 1)
00476     {
00477         builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex].ToString());
00478     }
00479     else
00480     {
00481         for (int j = selectionStart.CharacterIndex; j <= selectionEnd.CharacterIndex; j++)
00482         {
00483             builder.Append(this[selectionStart.BlockIndex][selectionStart.LineIndex][j].ToString());
00484         }
00485     }
00486 }
00487 }
00488
00489     return builder.ToString();
00490 }
00491
00492 /// <summary>
00493 /// Searches for the specified <see cref="Regex"/> in the text of the page. A single match
cannot span multiple lines.
00494 /// </summary>
00495 /// <param name="needle">The <see cref="Regex"/> to search for.</param>
00496 /// <returns>A lazy collection of <see cref="MuPDFStructuredTextAddressSpan"/>s representing
all the occurrences of the <paramref name="needle"/> in the text.</returns>
00497 public IEnumerable<MuPDFStructuredTextAddressSpan> Search(Regex needle)
00498 {
00499
00500     for (int i = 0; i < this.Count; i++)
00501     {
00502         if (this[i].Type == MuPDFStructuredTextBlock.Types.Text)
00503         {
00504             for (int j = 0; j < this[i].Count; j++)
00505             {
00506                 foreach (Match match in needle.Matches(this[i][j].Text))
00507                 {
00508                     if (match.Success)
00509                     {
00510                         int startIndex = 0;
00511                         int kStart = 0;
00512
00513                         while (startIndex < match.Index)
00514                         {
00515                             startIndex += this[i][j][kStart].Character.Length;
00516                             kStart++;
00517                         }
00518
00519                         if (startIndex > match.Index)
00520                         {
00521                             kStart--;
00522                         }
00523
00524                         int length = 0;
00525                         int kEnd = kStart - 1;
00526
00527                         while (length < match.Length)
00528                         {
00529                             kEnd++;
00530                             length += this[i][j][kEnd].Character.Length;
00531                         }
00532
00533                     }
00534                 }
00535             }
00536         }
00537     }
00538 }
```

```

00532                     yield return new MuPDFStructuredTextAddressSpan(new
00533                         MuPDFStructuredTextAddress(i, j, kStart), new MuPDFStructuredTextAddress(i, j, kEnd));
00534                     }
00535                 }
00536             }
00537         }
00538     }
00539 }
00540
00541 /// <inheritdoc>
00542 public IEnumarator<MuPDFStructuredTextBlock> GetEnumerator()
00543 {
00544     return ((IEnumerable<MuPDFStructuredTextBlock>)StructuredTextBlocks).GetEnumerator();
00545 }
00546
00547 IEnumerable IEnumerable.GetEnumerator()
00548 {
00549     return StructuredTextBlocks.GetEnumerator();
00550 }
00551 }
00552
00553 /// <summary>
00554 /// Represents a structured text block containing text or an image.
00555 /// </summary>
00556 public abstract class MuPDFStructuredTextBlock : IReadOnlyList<MuPDFStructuredTextLine>
00557 {
00558     /// <summary>
00559     /// Defines the type of the block.
00560     /// </summary>
00561     public enum Types
00562     {
00563         /// <summary>
00564         /// The block contains text.
00565         /// </summary>
00566         Text = 0,
00567
00568         /// <summary>
00569         /// The block contains an image.
00570         /// </summary>
00571         Image = 1
00572     }
00573
00574     /// <summary>
00575     /// The type of the block.
00576     /// </summary>
00577     public abstract Types Type { get; }
00578
00579     /// <summary>
00580     /// The bounding box of the block.
00581     /// </summary>
00582     public Rectangle BoundingBox { get; }
00583
00584     /// <summary>
00585     /// The number of lines in the block.
00586     /// </summary>
00587     public abstract int Count { get; }
00588
00589     /// <summary>
00590     /// Gets the specified line from the block.
00591     /// </summary>
00592     /// <param name="index">The index of the line to extract.</param>
00593     /// <returns>The <see cref="MuPDFStructuredTextLine"/> with the specified <paramref
00594     name="index"/>.</returns>
00595     public abstract MuPDFStructuredTextLine this[int index] { get; }
00596
00597     internal MuPDFStructuredTextBlock() { }
00598
00599     internal MuPDFStructuredTextBlock(Rectangle boundingBox)
00600     {
00601         this.BoundingBox = boundingBox;
00602     }
00603
00604     /// <inheritdoc>
00605     public abstract IEnumarator<MuPDFStructuredTextLine> GetEnumerator();
00606
00607     IEnumerable IEnumerable.GetEnumerator()
00608     {
00609         return this.GetEnumerator();
00610     }
00611
00612     /// <summary>
00613     /// Represents a block containing a single image. The block contains a single line with a single
00614     /// character.
00615     /// </summary>
00616     public class MuPDFImageStructuredTextBlock : MuPDFStructuredTextBlock

```

```

00616     {
00617         /// <inheritdoc/>
00618         public override Types.Type => Types.Image;
00619
00620         /// <inheritdoc/>
00621         public override int Count => 1;
00622
00623         private readonly MuPDFStructuredTextLine Line;
00624
00625         /// <inheritdoc/>
00626         public override MuPDFStructuredTextLine this[int index]
00627         {
00628             get
00629             {
00630                 if (index == 0)
00631                 {
00632                     return Line;
00633                 }
00634                 else
00635                 {
00636                     throw new IndexOutOfRangeException("A structured text block containing an image
only has one line!");
00637                 }
00638             }
00639         }
00640
00641         internal MuPDFImageStructuredTextBlock(Rectangle boundingBox) : base(boundingBox)
00642         {
00643             this.Line = new MuPDFStructuredTextLine(this.BoundingBox);
00644         }
00645
00646         /// <inheritdoc/>
00647         public override IEnumerator<MuPDFStructuredTextLine> GetEnumerator()
00648         {
00649             return ((IEnumerable<MuPDFStructuredTextLine>)new MuPDFStructuredTextLine[] { Line
}).GetEnumerator();
00650         }
00651     }
00652
00653     /// <summary>
00654     /// Represents a block containing multiple lines of text (typically a paragraph).
00655     /// </summary>
00656     public class MuPDFTextStructuredTextBlock : MuPDFStructuredTextBlock
00657     {
00658         /// <inheritdoc/>
00659         public override Types.Type => Types.Text;
00660
00661         /// <summary>
00662         /// The lines of text in the block.
00663         /// </summary>
00664         public MuPDFStructuredTextLine[] Lines { get; }
00665
00666         /// <inheritdoc/>
00667         public override int Count => ((IReadOnlyCollection<MuPDFStructuredTextLine>)Lines).Count;
00668
00669         /// <inheritdoc/>
00670         public override MuPDFStructuredTextLine this[int index] =>
((IReadOnlyList<MuPDFStructuredTextLine>)Lines)[index];
00671
00672         internal MuPDFTextStructuredTextBlock(Rectangle boundingBox, IntPtr blockPointer, int
lineCount) : base(boundingBox)
00673         {
00674             IntPtr[] linePointers = new IntPtr[lineCount];
00675             GCHandle linesHandle = GCHandle.Alloc(linePointers, GCHandleType.Pinned);
00676
00677             try
00678             {
00679                 ExitCodes result = (ExitCodes)NativeMethods.GetStructuredTextLines(blockPointer,
linesHandle.AddrOfPinnedObject());
00680
00681                 switch (result)
00682                 {
00683                     case ExitCodes.EXIT_SUCCESS:
00684                         break;
00685                     default:
00686                         throw new MuPDFException("Unknown error", result);
00687                 }
00688
00689                 Lines = new MuPDFStructuredTextLine[lineCount];
00690
00691                 for (int i = 0; i < lineCount; i++)
00692                 {
00693                     int wmode = -1;
00694                     float x0 = -1;
00695                     float y0 = -1;
00696                     float x1 = -1;
00697                     float y1 = -1;

```

```

00698                     float x = -1;
00699                     float y = -1;
00700
00701                     int charCount = -1;
00702
00703                     result = (ExitCodes)NativeMethods.GetStructuredTextLine(linePointers[i], ref
00704 wmode, ref x0, ref y0, ref xl, ref yl, ref x, ref y, ref charCount);
00705
00706                     switch (result)
00707                     {
00708                         case ExitCodes.EXIT_SUCCESS:
00709                             break;
00710                         default:
00711                             throw new MuPDFException("Unknown error", result);
00712                     }
00713
00714                     Rectangle bBox = new Rectangle(x0, y0, xl, yl);
00715                     PointF direction = new PointF(x, y);
00716
00717                     Lines[i] = new MuPDFStructuredTextLine(linePointers[i],
00718 (MuPDFStructuredTextLine.WritingModes)wmode, direction, bBox, charCount);
00719                 }
00720             }
00721         }
00722         linesHandle.Free();
00723     }
00724 }
00725
00726 /// <inheritdoc>
00727 public override IEnumerator<MuPDFStructuredTextLine> GetEnumerator()
00728 {
00729     return ((IEnumerable<MuPDFStructuredTextLine>)Lines).GetEnumerator();
00730 }
00731
00732 /// <summary>
00733 /// Returns the text contained in the block as a <see cref="string"/>.
00734 /// </summary>
00735 /// <returns>The text contained in the block as a <see cref="string"/>. If the block contains
00736 at least one line, the return value has a line terminator at the end.</returns>
00737 public override string ToString()
00738 {
00739     StringBuilder text = new StringBuilder();
00740
00741     foreach (MuPDFStructuredTextLine line in this)
00742     {
00743         text.AppendLine(line.Text);
00744     }
00745
00746     return text.ToString();
00747 }
00748
00749 /// <summary>
00750 /// Represents a single line of text (i.e. characters that share a common baseline).
00751 /// </summary>
00752 public class MuPDFStructuredTextLine : IReadOnlyList<MuPDFStructuredTextCharacter>
00753 {
00754     /// <summary>
00755     /// Defines the writing mode of the text.
00756     /// </summary>
00757     public enum WritingModes
00758     {
00759         /// <summary>
00760         /// The text is written horizontally.
00761         /// </summary>
00762         Horizontal = 0,
00763
00764         /// <summary>
00765         /// The text is written vertically.
00766         /// </summary>
00767         Vertical = 1
00768     }
00769
00770     /// <summary>
00771     /// The writing mode of the text.
00772     /// </summary>
00773     public WritingModes WritingMode { get; }
00774
00775     /// <summary>
00776     /// The normalised direction of the text baseline.
00777     /// </summary>
00778     public PointF Direction { get; }
00779
00780     /// <summary>
00781     /// The bounding box of the line.

```

```

00782     /// </summary>
00783     public Rectangle BoundingBox { get; }
00784
00785     /// <summary>
00786     /// The characters contained in the line.
00787     /// </summary>
00788     public MuPDFStructuredTextCharacter[] Characters { get; }
00789
00790     /// <summary>
00791     /// A string representation of the characters contained in the line.
00792     /// </summary>
00793     public string Text { get; }
00794
00795     /// <summary>
00796     /// The number of characters in the line.
00797     /// </summary>
00798     public int Count => ((IReadOnlyCollection<MuPDFStructuredTextCharacter>)Characters).Count;
00799
00800     /// <summary>
00801     /// Gets the specified character from the line.
00802     /// </summary>
00803     /// <param name="index">The index of the character.</param>
00804     /// <returns>The <see cref="MuPDFStructuredTextCharacter"/> with the specified <paramref
00805     name="index"/>.</returns>
00806     public MuPDFStructuredTextCharacter this[int index] =>
00807         ((IReadOnlyList<MuPDFStructuredTextCharacter>)Characters)[index];
00808
00809     internal MuPDFStructuredTextLine(Rectangle boundingBox)
00810     {
00811         this.BoundingBox = boundingBox;
00812         this.Characters = new MuPDFStructuredTextCharacter[]
00813         {
00814             new MuPDFStructuredTextCharacter(0, -1, new PointF(boundingBox.X0, boundingBox.Y1),
00815             new Quad(new PointF(boundingBox.X0, boundingBox.Y1), new PointF(boundingBox.X0, boundingBox.Y0), new
00816             PointF(boundingBox.X1, boundingBox.Y0), new PointF(boundingBox.X1, boundingBox.Y1)), 9)
00817         };
00818     }
00819
00820     internal MuPDFStructuredTextLine(IntPtr linePointer, WritingModes writingMode, PointF
00821     direction, Rectangle boundingBox, int charCount)
00822     {
00823         this.WritingMode = writingMode;
00824         this.Direction = direction;
00825         this.BoundingBox = boundingBox;
00826
00827         IntPtr[] charPointers = new IntPtr[charCount];
00828         GCHandle charsHandle = GCHandle.Alloc(charPointers, GCHandleType.Pinned);
00829
00830         try
00831         {
00832             ExitCodes result = (ExitCodes)NativeMethods.GetStructuredTextChars(linePointer,
00833             charsHandle.AddrOfPinnedObject());
00834
00835             switch (result)
00836             {
00837                 case ExitCodes.EXIT_SUCCESS:
00838                     break;
00839                 default:
00840                     throw new MuPDFException("Unknown error", result);
00841             }
00842
00843             Characters = new MuPDFStructuredTextCharacter[charCount];
00844
00845             StringBuilder textBuilder = new StringBuilder(charCount);
00846
00847             for (int i = 0; i < charCount; i++)
00848             {
00849                 int c = -1;
00850                 int color = -1;
00851                 float originX = -1;
00852                 float originY = -1;
00853                 float size = -1;
00854                 float llX = -1;
00855                 float llY = -1;
00856                 float ulX = -1;
00857                 float ulY = -1;
00858                 float urX = -1;
00859                 float urY = -1;
00860                 float lrX = -1;
00861                 float lrY = -1;
00862
00863                 result = (ExitCodes)NativeMethods.GetStructuredTextChar(charPointers[i], ref c,
00864                 ref color, ref originX, ref originY, ref size, ref llX, ref llY, ref ulX, ref ulY, ref urX, ref urY,
00865                 ref lrX, ref lrY);
00866
00867                 switch (result)
00868                 {
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02510
02511
02512
02513
025
```

```

00861             case ExitCodes.EXIT_SUCCESS:
00862                 break;
00863             default:
00864                 throw new MuPDFException("Unknown error", result);
00865         }
00866     }
00867     Quad quad = new Quad(new PointF(llX, llY), new PointF(ulX, ulY), new PointF(urX,
00868     urY), new PointF(lrX, lrY));
00869     PointF origin = new PointF(originX, originY);
00870     Characters[i] = new MuPDFStructuredTextCharacter(c, color, origin, quad, size);
00871     textBuilder.Append(Characters[i].Character);
00872 }
00873
00874     this.Text = textBuilder.ToString();
00875 }
00876 finally
00877 {
00878     charsHandle.Free();
00879 }
00880 }
00881
00882 /// <summary>
00883 /// Returns a string representation of the line.
00884 /// </summary>
00885 /// <returns>A string representation of the line.</returns>
00886 public override string ToString()
00887 {
00888     return this.Text;
00889 }
00890
00891 /// <inheritDoc/>
00892 public IEnumarator<MuPDFStructuredTextCharacter> GetEnumerator()
00893 {
00894     return ((IEnumarable<MuPDFStructuredTextCharacter>)Characters).GetEnumarator();
00895 }
00896
00897 IEnumarator IEnumarable.GetEnumerator()
00898 {
00899     return Characters.GetEnumerator();
00900 }
00901 }
00902
00903 /// <summary>
00904 /// Represents a single text character.
00905 /// </summary>
00906 public class MuPDFStructuredTextCharacter
00907 {
00908     /// <summary>
00909     /// The unicode code point of the character.
00910     /// </summary>
00911     public int CodePoint { get; }
00912
00913     /// <summary>
00914     /// A string representation of the character. It may consist of a single <see cref="char"/> or
00915     /// of a surrogate pair of <see cref="char"/>s.
00916     /// </summary>
00917     public string Character { get; }
00918
00919     /// <summary>
00920     /// An sRGB hex representation of the colour of the character.
00921     /// </summary>
00922     public int Color { get; }
00923
00924     /// <summary>
00925     /// The baseline origin of the character.
00926     /// </summary>
00927     public PointF Origin { get; }
00928
00929     /// <summary>
00930     /// A quadrilater bound for the character. This may or may not be a rectangle.
00931     /// </summary>
00932     public Quad BoundingQuad { get; }
00933
00934     /// <summary>
00935     /// The size in points of the character.
00936     /// </summary>
00937     public float Size { get; }
00938
00939     internal MuPDFStructuredTextCharacter(int codePoint, int color, PointF origin, Quad
00940     boundingQuad, float size)
00941     {
00942         this.CodePoint = codePoint;
00943         this.Character = Char.ConvertFromUtf32(codePoint);
00944         this.Color = color;
00945         this.Origin = origin;
00946         this.BoundingQuad = boundingQuad;

```

```

00945         this.Size = size;
00946     }
00947
00948     /// <summary>
00949     /// Returns a string representation of the character.
00950     /// </summary>
00951     /// <returns>A string representation of the character.</returns>
00952     public override string ToString()
00953     {
00954         return this.Character;
00955     }
00956 }
00957
00958     /// <summary>
00959     /// Represents the address of a particular character in a <see cref="MuPDFStructuredTextPage"/>,
00960     /// in terms of block index, line index and character index.
00961     /// </summary>
00962     public struct MuPDFStructuredTextAddress : IComparable<MuPDFStructuredTextAddress>,
00963     IEquatable<MuPDFStructuredTextAddress>
00964     {
00965         /// <summary>
00966         /// The index of the block.
00967         /// </summary>
00968         public readonly int BlockIndex;
00969
00970         /// <summary>
00971         /// The index of the line within the block.
00972         /// </summary>
00973         public readonly int LineIndex;
00974
00975         /// <summary>
00976         /// The index of the character within the line.
00977         /// </summary>
00978         public readonly int CharacterIndex;
00979
00980         /// <summary>
00981         /// Creates a new <see cref="MuPDFStructuredTextAddress"/> from the specified indices.
00982         /// </summary>
00983         /// <param name="blockIndex">The index of the block.</param>
00984         /// <param name="lineIndex">The index of the line within the block.</param>
00985         /// <param name="characterIndex">The index of the character within the line.</param>
00986         public MuPDFStructuredTextAddress(int blockIndex, int lineIndex, int characterIndex)
00987         {
00988             this.BlockIndex = blockIndex;
00989             this.LineIndex = lineIndex;
00990             this.CharacterIndex = characterIndex;
00991
00992             /// <summary>
00993             /// Compares this <see cref="MuPDFStructuredTextAddress"/> with another <see
00994             /// cref="MuPDFStructuredTextAddress"/>.
00995             /// </summary>
00996             /// <param name="other">The <see cref="MuPDFStructuredTextAddress"/> to compare with the
00997             /// current instance.</param>
00998             /// <returns>-1 if the <paramref name="other"/> <see cref="MuPDFStructuredTextAddress"/> comes
00999             /// after the current instance, 1 if it comes before, or 0 if they represent the same address.</returns>
01000             public int CompareTo(MuPDFStructuredTextAddress other)
01001             {
01002                 if (this < other)
01003                 {
01004                     return -1;
01005                 }
01006                 else if (this > other)
01007                 {
01008                     return 1;
01009                 }
01010             }
01011
01012             /// <summary>
01013             /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01014             /// </summary>
01015             /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01016             /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to
01017             /// compare.</param>
01018             /// <returns><see langword="true"/> if the <paramref name="first"/> <see
01019             /// cref="MuPDFStructuredTextAddress"/> comes after the <paramref name="second"/> one; otherwise, <see
01020             /// langword="false"/>.</returns>
01021             public static bool operator >(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
01022             second)
01023             {
01024                 if (first.BlockIndex > second.BlockIndex)
01025                 {
01026                     return true;
01027                 }
01028             }
01029         }
01030     }
01031 }
```

```

01023         }
01024         else if (first.BlockIndex < second.BlockIndex)
01025     {
01026         return false;
01027     }
01028     else
01029     {
01030         if (first.LineIndex > second.LineIndex)
01031     {
01032         return true;
01033     }
01034     else if (first.LineIndex < second.LineIndex)
01035     {
01036         return false;
01037     }
01038     else
01039     {
01040         if (first.CharacterIndex > second.CharacterIndex)
01041     {
01042         return true;
01043     }
01044     else
01045     {
01046         return false;
01047     }
01048 }
01049 }
01050 }
01051
01052     /// <summary>
01053     /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01054     /// </summary>
01055     /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01056     /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to
01057     /// compare.</param>
01058     /// <returns><see langword="true"/> if the <paramref name="first"/> <see
01059     /// cref="MuPDFStructuredTextAddress"/> comes after the <paramref name="second"/> one or if they
01060     /// represent the same address; otherwise, <see langword="false"/>.</returns>
01061     public static bool operator >=(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
01062     second)
01063     {
01064         if (first.BlockIndex > second.BlockIndex)
01065     {
01066         return true;
01067     }
01068     else if (first.BlockIndex < second.BlockIndex)
01069     {
01070         return false;
01071     }
01072     else
01073     {
01074         if (first.LineIndex > second.LineIndex)
01075     {
01076         return true;
01077     }
01078     else if (first.LineIndex < second.LineIndex)
01079     {
01080         return false;
01081     }
01082     else
01083     {
01084         if (first.CharacterIndex >= second.CharacterIndex)
01085     {
01086         return true;
01087     }
01088     }
01089 }
01090 }
01091
01092     /// <summary>
01093     /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01094     /// </summary>
01095     /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01096     /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to
01097     /// compare.</param>
01098     /// <returns><see langword="true"/> if the <paramref name="first"/> <see
01099     /// cref="MuPDFStructuredTextAddress"/> comes before the <paramref name="second"/> one; otherwise, <see
01100     /// langword="false"/>.</returns>
01101     public static bool operator <(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
01102     second)
01103     {
01104         if (first.BlockIndex > second.BlockIndex)
01105     {

```

```
01102         return false;
01103     }
01104     else if (first.BlockIndex < second.BlockIndex)
01105     {
01106         return true;
01107     }
01108     else
01109     {
01110         if (first.LineIndex > second.LineIndex)
01111         {
01112             return false;
01113         }
01114         else if (first.LineIndex < second.LineIndex)
01115         {
01116             return true;
01117         }
01118         else
01119         {
01120             if (first.CharacterIndex < second.CharacterIndex)
01121             {
01122                 return true;
01123             }
01124             else
01125             {
01126                 return false;
01127             }
01128         }
01129     }
01130 }
01131
01132 /// <summary>
01133 /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01134 /// </summary>
01135 /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01136 /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to
01137 compare.</param>
01138     /// <returns><see langword="true"/> if the <paramref name="first"/> <see
01139 cref="MuPDFStructuredTextAddress"/> comes before the <paramref name="second"/> one or if they
01140 represent the same address; otherwise, <see langword="false"/>.</returns>
01141     public static bool operator <=(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
01142 second)
01143     {
01144         if (first.BlockIndex > second.BlockIndex)
01145         {
01146             return false;
01147         }
01148         else if (first.BlockIndex < second.BlockIndex)
01149         {
01150             return true;
01151         }
01152         else
01153         {
01154             if (first.LineIndex > second.LineIndex)
01155             {
01156                 return false;
01157             }
01158             else if (first.LineIndex < second.LineIndex)
01159             {
01160                 return true;
01161             }
01162             else
01163             {
01164                 if (first.CharacterIndex <= second.CharacterIndex)
01165                 {
01166                     return true;
01167                 }
01168             }
01169         }
01170     }
01171
01172 /// <summary>
01173 /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01174 /// </summary>
01175 /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01176 /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to
01177 compare.</param>
01178     /// <returns><see langword="true"/> if the two <see cref="MuPDFStructuredTextAddress"/>es
01179 represent the same address; otherwise, <see langword="false"/>.</returns>
01180     public static bool operator ==(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
01181 second)
01182     {
01183         return first.CharacterIndex == second.CharacterIndex && first.LineIndex ==
01184 second.LineIndex && first.BlockIndex == second.BlockIndex;
```

```

01181         }
01182
01183         /// <summary>
01184         /// Compares two <see cref="MuPDFStructuredTextAddress"/>.
01185         /// </summary>
01186         /// <param name="first">The first <see cref="MuPDFStructuredTextAddress"/> to compare.</param>
01187         /// <param name="second">The second <see cref="MuPDFStructuredTextAddress"/> to
01188         /// compare.</param>
01189         /// <returns><see langword="true"/> if the two <see cref="MuPDFStructuredTextAddress"/>es
01190         /// represent different addresses; otherwise, <see langword="false"/>.</returns>
01191         public static bool operator !=(MuPDFStructuredTextAddress first, MuPDFStructuredTextAddress
01192         second)
01193         {
01194             return first.CharacterIndex != second.CharacterIndex || first.LineIndex !=
01195             second.LineIndex || first.BlockIndex != second.BlockIndex;
01196         }
01197
01198         /// <inheritdoc/>
01199         public override int GetHashCode()
01200         {
01201             unchecked
01202             {
01203                 return ((this.BlockIndex * 33 * 33) ^ this.LineIndex * 33) ^ this.CharacterIndex;
01204             }
01205
01206             /// <summary>
01207             /// Returns a <see cref="MuPDFStructuredTextAddress"/> corresponding to the next character in
01208             /// the specified page.
01209             /// </summary>
01210             /// <param name="page">The page the address refers to.</param>
01211             /// <returns>A <see cref="MuPDFStructuredTextAddress"/> corresponding to the next character in
01212             /// the specified page.</returns>
01213             public MuPDFStructuredTextAddress? Increment(MuPDFStructuredTextPage page)
01214             {
01215                 int newBlockIndex = this.BlockIndex;
01216                 int newLineIndex = this.LineIndex;
01217                 int newCharacterIndex = this.CharacterIndex + 1;
01218
01219                 if (page[newBlockIndex] [newLineIndex].Count <= newCharacterIndex)
01220                 {
01221                     newCharacterIndex = 0;
01222                     newLineIndex++;
01223                 }
01224
01225                 if (page[newBlockIndex].Count <= newLineIndex)
01226                 {
01227                     newLineIndex = 0;
01228                     newBlockIndex++;
01229                 }
01230
01231                 if (page.Count <= newBlockIndex)
01232                 {
01233                     return null;
01234                 }
01235
01236                 return new MuPDFStructuredTextAddress(newBlockIndex, newLineIndex, newCharacterIndex);
01237             }
01238
01239             /// <summary>
01240             /// Compares the current <see cref="MuPDFStructuredTextAddress"/> with another <see
01241             /// cref="MuPDFStructuredTextAddress"/>.
01242             /// </summary>
01243             /// <param name="other">The other <see cref="MuPDFStructuredTextAddress"/> to compare with the
01244             /// current instance.</param>
01245             /// <returns><see langword="true"/> if the two <see cref="MuPDFStructuredTextAddress"/>es
01246             /// represent the same address; otherwise, <see langword="false"/>.</returns>
01247             public bool Equals(MuPDFStructuredTextAddress other)
01248             {
01249                 return this.CharacterIndex == other.CharacterIndex && this.LineIndex == other.LineIndex &&
01250                 this.BlockIndex == other.BlockIndex;
01251
01252             /// <summary>
01253             /// Represents a range of characters in a <see cref="MuPDFStructuredTextPage"/>.
01254             /// </summary>
01255             public class MuPDFStructuredTextAddressSpan
01256             {
01257                 /// <summary>
01258                 /// The address of the start of the range.

```

```

01258     /// </summary>
01259     public readonly MuPDFStructuredTextAddress Start;
01260
01261     /// <summary>
01262     /// The address of the end of the range (inclusive), or <see langword="null" /> to signify an
01263     /// empty range.
01264     public readonly MuPDFStructuredTextAddress? End;
01265
01266     /// <summary>
01267     /// Creates a new <see cref="MuPDFStructuredTextAddressSpan"/> corresponding to the specified
01268     /// character range.
01269     /// <param name="start">The address of the start of the range.</param>
01270     /// <param name="end">The address of the end of the range (inclusive), or <see langword="null" />
01271     /// to signify an empty range.</param>
01272     public MuPDFStructuredTextAddressSpan(MuPDFStructuredTextAddress start,
01273                                         MuPDFStructuredTextAddress? end)
01274     {
01275         this.Start = start;
01276         this.End = end;
01277     }

```

8.11 Rectangles.cs

```

00001 /*
00002  MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
00003  Copyright (C) 2020 Giorgio Bianchini
00004
00005  This program is free software: you can redistribute it and/or modify
00006  it under the terms of the GNU Affero General Public License as
00007  published by the Free Software Foundation, version 3.
00008
00009  This program is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU Affero General Public License for more details.
00013
00014  You should have received a copy of the GNU Affero General Public License
00015  along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using System;
00019
00020 namespace MuPDFCore
00021 {
00022     /// <summary>
00023     /// Represents the size of a rectangle.
00024     /// </summary>
00025     public struct Size
00026     {
00027         /// <summary>
00028         /// The width of the rectangle.
00029         /// </summary>
00030         public float Width;
00031
00032         /// <summary>
00033         /// The height of the rectangle.
00034         /// </summary>
00035         public float Height;
00036
00037         /// <summary>
00038         /// Create a new <see cref="Size"/> with the specified width and height.
00039         /// </summary>
00040         /// <param name="width">The width of the rectangle.</param>
00041         /// <param name="height">The height of the rectangle.</param>
00042         public Size(float width, float height)
00043         {
00044             Width = width;
00045             Height = height;
00046         }
00047
00048         /// <summary>
00049         /// Create a new <see cref="Size"/> with the specified width and height.
00050         /// </summary>
00051         /// <param name="width">The width of the rectangle.</param>
00052         /// <param name="height">The height of the rectangle.</param>
00053         public Size(double width, double height)
00054         {
00055             Width = (float)width;
00056             Height = (float)height;

```

```

00057         }
00058
00059     /// <summary>
00060     /// Split the size into the specified number of <see cref="Rectangle"/>s.
00061     /// </summary>
00062     /// <param name="divisions">The number of rectangles in which the size should be split. This
00063     /// must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than
00064     /// <paramref name="divisions"/> that satisfies this condition is used.</param>
00065     /// <returns>An array of <see cref="Rectangle"/>s that when positioned properly cover an area
00066     /// of the size of this object.</returns>
00067     public Rectangle[] Split(int divisions)
00068     {
00069         divisions = Utils.GetAcceptableNumber(divisions);
00070
00071         Rectangle[] tbr = new Rectangle[divisions];
00072
00073         bool isVertical = this.Height > this.Width;
00074
00075         if (divisions == 1)
00076         {
00077             tbr[0] = new Rectangle(0, 0, Width, Height);
00078         }
00079         else if (divisions == 2)
00080         {
00081             if (isVertical)
00082             {
00083                 tbr[0] = new Rectangle(0, 0, Width, Height / 2);
00084                 tbr[1] = new Rectangle(0, Height / 2, Width, Height);
00085             }
00086             else
00087             {
00088                 tbr[0] = new Rectangle(0, 0, Width / 2, Height);
00089                 tbr[1] = new Rectangle(Width / 2, 0, Width, Height);
00090             }
00091         }
00092         else if (divisions == 3)
00093         {
00094             if (isVertical)
00095             {
00096                 tbr[0] = new Rectangle(0, 0, Width / 2, 2 * Height / 3);
00097                 tbr[1] = new Rectangle(Width / 2, 0, Width, 2 * Height / 3);
00098                 tbr[2] = new Rectangle(0, 2 * Height / 3, Width, Height);
00099             }
00100             else
00101             {
00102                 tbr[0] = new Rectangle(0, 0, 2 * Width / 3, Height / 2);
00103                 tbr[1] = new Rectangle(0, Height / 2, 2 * Width / 3, Height);
00104                 tbr[2] = new Rectangle(2 * Width / 3, 0, Width, Height);
00105             }
00106         }
00107         else if (divisions == 5)
00108         {
00109             if (isVertical)
00110             {
00111                 tbr[0] = new Rectangle(0, 0, Width / 2, 2 * Height / 5);
00112                 tbr[1] = new Rectangle(Width / 2, 0, Width, 2 * Height / 5);
00113                 tbr[2] = new Rectangle(0, 2 * Height / 5, Width / 2, 4 * Height / 5);
00114                 tbr[3] = new Rectangle(Width / 2, 2 * Height / 5, Width, 4 * Height / 5);
00115                 tbr[4] = new Rectangle(0, 4 * Height / 5, Width, Height);
00116             }
00117             else
00118             {
00119                 tbr[0] = new Rectangle(0, 0, 2 * Width / 5, Height / 2);
00120                 tbr[1] = new Rectangle(0, Height / 2, 2 * Width / 5, Height);
00121                 tbr[2] = new Rectangle(2 * Width / 5, 0, 4 * Width / 5, Height / 2);
00122                 tbr[3] = new Rectangle(2 * Width / 5, Height / 2, 4 * Width / 5, Height);
00123                 tbr[4] = new Rectangle(4 * Width / 5, 0, Width, Height);
00124             }
00125         }
00126         else if (divisions == 7)
00127         {
00128             if (isVertical)
00129             {
00130                 tbr[0] = new Rectangle(0, 0, Width / 2, 2 * Height / 7);
00131                 tbr[1] = new Rectangle(Width / 2, 0, Width, 2 * Height / 7);
00132                 tbr[2] = new Rectangle(0, 2 * Height / 7, Width / 2, 4 * Height / 7);
00133                 tbr[3] = new Rectangle(Width / 2, 2 * Height / 7, Width, 4 * Height / 7);
00134                 tbr[4] = new Rectangle(0, 4 * Height / 7, Width / 2, 6 * Height / 7);
00135                 tbr[5] = new Rectangle(Width / 2, 4 * Height / 7, Width, 6 * Height / 7);
00136                 tbr[6] = new Rectangle(0, 6 * Height / 7, Width, Height);
00137             }
00138             else
00139             {
00140                 tbr[0] = new Rectangle(0, 0, 2 * Width / 7, Height / 2);
00141                 tbr[1] = new Rectangle(0, Height / 2, 2 * Width / 7, Height);
00142                 tbr[2] = new Rectangle(2 * Width / 7, 0, 4 * Width / 7, Height / 2);
00143                 tbr[3] = new Rectangle(2 * Width / 7, Height / 2, 4 * Width / 7, Height);
00144             }
00145         }
00146     }
00147 }
```

```

00141             tbr[4] = new Rectangle(4 * Width / 7, 0, 6 * Width / 7, Height / 2);
00142             tbr[5] = new Rectangle(4 * Width / 7, Height / 2, 6 * Width / 7, Height);
00143             tbr[6] = new Rectangle(6 * Width / 7, 0, Width, Height);
00144         }
00145     }
00146     else
00147     {
00148         for (int divisorInd = 0; divisorInd < Utils.AcceptableDivisors.Length; divisorInd++)
00149         {
00150             if (divisions % Utils.AcceptableDivisors[divisorInd] == 0)
00151             {
00152                 Rectangle[] largerDivisions = this.Split(divisions /
00153                     Utils.AcceptableDivisors[divisorInd]);
00154
00155                 int pos = 0;
00156
00157                 for (int i = 0; i < largerDivisions.Length; i++)
00158                 {
00159                     Size s = new Size(largerDivisions[i].Width, largerDivisions[i].Height);
00160                     Rectangle[] currDivision = s.Split(Utils.AcceptableDivisors[divisorInd]);
00161
00162                     for (int j = 0; j < currDivision.Length; j++)
00163                     {
00164                         tbr[pos] = new Rectangle(largerDivisions[i].X0 + currDivision[j].X0,
00165                             largerDivisions[i].Y0 + currDivision[j].Y0, largerDivisions[i].X0 + currDivision[j].X1,
00166                             largerDivisions[i].Y0 + currDivision[j].Y1);
00167                         pos++;
00168                     }
00169                 }
00170             }
00171         }
00172
00173         return tbr;
00174     }
00175 }
00176 }
00177
00178 /// <summary>
00179 /// Represents the size of a rectangle using only integer numbers.
00180 /// </summary>
00181 public struct RoundedSize
00182 {
00183     /// <summary>
00184     /// The width of the rectangle.
00185     /// </summary>
00186     public int Width;
00187
00188     /// <summary>
00189     /// The height of the rectangle.
00190     /// </summary>
00191     public int Height;
00192
00193     /// <summary>
00194     /// Create a new <see cref="RoundedSize"/> with the specified width and height.
00195     /// </summary>
00196     /// <param name="width">The width of the rectangle.</param>
00197     /// <param name="height">The height of the rectangle.</param>
00198     public RoundedSize(int width, int height)
00199     {
00200         Width = width;
00201         Height = height;
00202     }
00203
00204     /// <summary>
00205     /// Split the size into the specified number of <see cref="RoundedRectangle"/>s.
00206     /// </summary>
00207     /// <param name="divisions">The number of rectangles in which the size should be split. This
00208     must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller than
00209     <paramref name="divisions"/> that satisfies this condition is used.</param>
00210     /// <returns>An array of <see cref="RoundedRectangle"/>s that when positioned properly cover
00211     an area of the size of this object.</returns>
00212     public RoundedRectangle[] Split(int divisions)
00213     {
00214         divisions = Utils.GetAcceptableNumber(divisions);
00215
00216         RoundedRectangle[] tbr = new RoundedRectangle[divisions];
00217
00218         bool isVertical = this.Height > this.Width;
00219
00220         if (divisions == 1)
00221         {
00222             tbr[0] = new RoundedRectangle(0, 0, Width, Height);
00223         }
00224         else if (divisions == 2)
00225         {
00226             tbr[0] = new RoundedRectangle(0, 0, Width / 2, Height);
00227             tbr[1] = new RoundedRectangle(Width / 2, 0, Width, Height);
00228         }
00229         else if (divisions == 3)
00230         {
00231             tbr[0] = new RoundedRectangle(0, 0, Width / 3, Height);
00232             tbr[1] = new RoundedRectangle(Width / 3, 0, 2 * Width / 3, Height);
00233             tbr[2] = new RoundedRectangle(2 * Width / 3, 0, Width, Height);
00234         }
00235         else if (divisions == 4)
00236         {
00237             tbr[0] = new RoundedRectangle(0, 0, Width / 4, Height);
00238             tbr[1] = new RoundedRectangle(Width / 4, 0, 3 * Width / 8, Height);
00239             tbr[2] = new RoundedRectangle(3 * Width / 8, 0, 5 * Width / 8, Height);
00240             tbr[3] = new RoundedRectangle(5 * Width / 8, 0, Width, Height);
00241         }
00242         else if (divisions == 5)
00243         {
00244             tbr[0] = new RoundedRectangle(0, 0, Width / 5, Height);
00245             tbr[1] = new RoundedRectangle(Width / 5, 0, 4 * Width / 25, Height);
00246             tbr[2] = new RoundedRectangle(4 * Width / 25, 0, 11 * Width / 25, Height);
00247             tbr[3] = new RoundedRectangle(11 * Width / 25, 0, 16 * Width / 25, Height);
00248             tbr[4] = new RoundedRectangle(16 * Width / 25, 0, Width, Height);
00249         }
00250         else if (divisions == 6)
00251         {
00252             tbr[0] = new RoundedRectangle(0, 0, Width / 6, Height);
00253             tbr[1] = new RoundedRectangle(Width / 6, 0, 5 * Width / 36, Height);
00254             tbr[2] = new RoundedRectangle(5 * Width / 36, 0, 17 * Width / 36, Height);
00255             tbr[3] = new RoundedRectangle(17 * Width / 36, 0, 23 * Width / 36, Height);
00256             tbr[4] = new RoundedRectangle(23 * Width / 36, 0, 29 * Width / 36, Height);
00257             tbr[5] = new RoundedRectangle(29 * Width / 36, 0, Width, Height);
00258         }
00259         else if (divisions == 7)
00260         {
00261             tbr[0] = new RoundedRectangle(0, 0, Width / 7, Height);
00262             tbr[1] = new RoundedRectangle(Width / 7, 0, 6 * Width / 49, Height);
00263             tbr[2] = new RoundedRectangle(6 * Width / 49, 0, 22 * Width / 49, Height);
00264             tbr[3] = new RoundedRectangle(22 * Width / 49, 0, 38 * Width / 49, Height);
00265             tbr[4] = new RoundedRectangle(38 * Width / 49, 0, 44 * Width / 49, Height);
00266             tbr[5] = new RoundedRectangle(44 * Width / 49, 0, 50 * Width / 49, Height);
00267             tbr[6] = new RoundedRectangle(50 * Width / 49, 0, Width, Height);
00268         }
00269         else if (divisions == 8)
00270         {
00271             tbr[0] = new RoundedRectangle(0, 0, Width / 8, Height);
00272             tbr[1] = new RoundedRectangle(Width / 8, 0, 7 * Width / 64, Height);
00273             tbr[2] = new RoundedRectangle(7 * Width / 64, 0, 15 * Width / 64, Height);
00274             tbr[3] = new RoundedRectangle(15 * Width / 64, 0, 23 * Width / 64, Height);
00275             tbr[4] = new RoundedRectangle(23 * Width / 64, 0, 31 * Width / 64, Height);
00276             tbr[5] = new RoundedRectangle(31 * Width / 64, 0, 39 * Width / 64, Height);
00277             tbr[6] = new RoundedRectangle(39 * Width / 64, 0, 47 * Width / 64, Height);
00278             tbr[7] = new RoundedRectangle(47 * Width / 64, 0, Width, Height);
00279         }
00280         else if (divisions == 9)
00281         {
00282             tbr[0] = new RoundedRectangle(0, 0, Width / 9, Height);
00283             tbr[1] = new RoundedRectangle(Width / 9, 0, 8 * Width / 81, Height);
00284             tbr[2] = new RoundedRectangle(8 * Width / 81, 0, 17 * Width / 81, Height);
00285             tbr[3] = new RoundedRectangle(17 * Width / 81, 0, 26 * Width / 81, Height);
00286             tbr[4] = new RoundedRectangle(26 * Width / 81, 0, 35 * Width / 81, Height);
00287             tbr[5] = new RoundedRectangle(35 * Width / 81, 0, 44 * Width / 81, Height);
00288             tbr[6] = new RoundedRectangle(44 * Width / 81, 0, 53 * Width / 81, Height);
00289             tbr[7] = new RoundedRectangle(53 * Width / 81, 0, 62 * Width / 81, Height);
00290             tbr[8] = new RoundedRectangle(62 * Width / 81, 0, Width, Height);
00291         }
00292         else if (divisions == 10)
00293         {
00294             tbr[0] = new RoundedRectangle(0, 0, Width / 10, Height);
00295             tbr[1] = new RoundedRectangle(Width / 10, 0, 9 * Width / 100, Height);
00296             tbr[2] = new RoundedRectangle(9 * Width / 100, 0, 19 * Width / 100, Height);
00297             tbr[3] = new RoundedRectangle(19 * Width / 100, 0, 29 * Width / 100, Height);
00298             tbr[4] = new RoundedRectangle(29 * Width / 100, 0, 39 * Width / 100, Height);
00299             tbr[5] = new RoundedRectangle(39 * Width / 100, 0, 49 * Width / 100, Height);
00300             tbr[6] = new RoundedRectangle(49 * Width / 100, 0, 59 * Width / 100, Height);
00301             tbr[7] = new RoundedRectangle(59 * Width / 100, 0, 69 * Width / 100, Height);
00302             tbr[8] = new RoundedRectangle(69 * Width / 100, 0, 79 * Width / 100, Height);
00303             tbr[9] = new RoundedRectangle(79 * Width / 100, 0, Width, Height);
00304         }
00305         else if (divisions == 11)
00306         {
00307             tbr[0] = new RoundedRectangle(0, 0, Width / 11, Height);
00308             tbr[1] = new RoundedRectangle(Width / 11, 0, 10 * Width / 121, Height);
00309             tbr[2] = new RoundedRectangle(10 * Width / 121, 0, 21 * Width / 121, Height);
00310             tbr[3] = new RoundedRectangle(21 * Width / 121, 0, 32 * Width / 121, Height);
00311             tbr[4] = new RoundedRectangle(32 * Width / 121, 0, 43 * Width / 121, Height);
00312             tbr[5] = new RoundedRectangle(43 * Width / 121, 0, 54 * Width / 121, Height);
00313             tbr[6] = new RoundedRectangle(54 * Width / 121, 0, 65 * Width / 121, Height);
00314             tbr[7] = new RoundedRectangle(65 * Width / 121, 0, 76 * Width / 121, Height);
00315             tbr[8] = new RoundedRectangle(76 * Width / 121, 0, 87 * Width / 121, Height);
00316             tbr[9] = new RoundedRectangle(87 * Width / 121, 0, 98 * Width / 121, Height);
00317             tbr[10] = new RoundedRectangle(98 * Width / 121, 0, Width, Height);
00318         }
00319         else if (divisions == 12)
00320         {
00321             tbr[0] = new RoundedRectangle(0, 0, Width / 12, Height);
00322             tbr[1] = new RoundedRectangle(Width / 12, 0, 11 * Width / 144, Height);
00323             tbr[2] = new RoundedRectangle(11 * Width / 144, 0, 23 * Width / 144, Height);
00324             tbr[3] = new RoundedRectangle(23 * Width / 144, 0, 35 * Width / 144, Height);
00325             tbr[4] = new RoundedRectangle(35 * Width / 144, 0, 47 * Width / 144, Height);
00326             tbr[5] = new RoundedRectangle(47 * Width / 144, 0, 59 * Width / 144, Height);
00327             tbr[6] = new RoundedRectangle(59 * Width / 144, 0, 71 * Width / 144, Height);
00328             tbr[7] = new RoundedRectangle(71 * Width / 144, 0, 83 * Width / 144, Height);
00329             tbr[8] = new RoundedRectangle(83 * Width / 144, 0, 95 * Width / 144, Height);
00330             tbr[9] = new RoundedRectangle(95 * Width / 144, 0, 107 * Width / 144, Height);
00331             tbr[10] = new RoundedRectangle(107 * Width / 144, 0, 119 * Width / 144, Height);
00332             tbr[11] = new RoundedRectangle(119 * Width / 144, 0, Width, Height);
00333         }
00334         else if (divisions == 13)
00335         {
00336             tbr[0] = new RoundedRectangle(0, 0, Width / 13, Height);
00337             tbr[1] = new RoundedRectangle(Width / 13, 0, 12 * Width / 169, Height);
00338             tbr[2] = new RoundedRectangle(12 * Width / 169, 0, 25 * Width / 169, Height);
00339             tbr[3] = new RoundedRectangle(25 * Width / 169, 0, 38 * Width / 169, Height);
00340             tbr[4] = new RoundedRectangle(38 * Width / 169, 0, 51 * Width / 169, Height);
00341             tbr[5] = new RoundedRectangle(51 * Width / 169, 0, 64 * Width / 169, Height);
00342             tbr[6] = new RoundedRectangle(64 * Width / 169, 0, 77 * Width / 169, Height);
00343             tbr[7] = new RoundedRectangle(77 * Width / 169, 0, 90 * Width / 169, Height);
00344             tbr[8] = new RoundedRectangle(90 * Width / 169, 0, 103 * Width / 169, Height);
00345             tbr[9] = new RoundedRectangle(103 * Width / 169, 0, 116 * Width / 169, Height);
00346             tbr[10] = new RoundedRectangle(116 * Width / 169, 0, 129 * Width / 169, Height);
00347             tbr[11] = new RoundedRectangle(129 * Width / 169, 0, 142 * Width / 169, Height);
00348             tbr[12] = new RoundedRectangle(142 * Width / 169, 0, Width, Height);
00349         }
00350         else if (divisions == 14)
00351         {
00352             tbr[0] = new RoundedRectangle(0, 0, Width / 14, Height);
00353             tbr[1] = new RoundedRectangle(Width / 14, 0, 13 * Width / 196, Height);
00354             tbr[2] = new RoundedRectangle(13 * Width / 196, 0, 27 * Width / 196, Height);
00355             tbr[3] = new RoundedRectangle(27 * Width / 196, 0, 41 * Width / 196, Height);
00356             tbr[4] = new RoundedRectangle(41 * Width / 196, 0, 55 * Width / 196, Height);
00357             tbr[5] = new RoundedRectangle(55 * Width / 196, 0, 69 * Width / 196, Height);
00358             tbr[6] = new RoundedRectangle(69 * Width / 196, 0, 83 * Width / 196, Height);
00359             tbr[7] = new RoundedRectangle(83 * Width / 196, 0, 97 * Width / 196, Height);
00360             tbr[8] = new RoundedRectangle(97 * Width / 196, 0, 111 * Width / 196, Height);
00361             tbr[9] = new RoundedRectangle(111 * Width / 196, 0, 125 * Width / 196, Height);
00362             tbr[10] = new RoundedRectangle(125 * Width / 196, 0, 139 * Width / 196, Height);
00363             tbr[11] = new RoundedRectangle(139 * Width / 196, 0, 153 * Width / 196, Height);
00364             tbr[12] = new RoundedRectangle(153 * Width / 196, 0, 167 * Width / 196, Height);
00365             tbr[13] = new RoundedRectangle(167 * Width / 196, 0, Width, Height);
00366         }
00367         else if (divisions == 15)
00368         {
00369             tbr[0] = new RoundedRectangle(0, 0, Width / 15, Height);
00370             tbr[1] = new RoundedRectangle(Width / 15, 0, 14 * Width / 225, Height);
00371             tbr[2] = new RoundedRectangle(14 * Width / 225, 0, 29 * Width / 225, Height);
00372             tbr[3] = new RoundedRectangle(29 * Width / 225, 0, 44 * Width / 225, Height);
00373             tbr[4] = new RoundedRectangle(44 * Width / 225, 0, 59 * Width / 225, Height);
00374             tbr[5] = new RoundedRectangle(59 * Width / 225, 0, 74 * Width / 225, Height);
00375             tbr[6] = new RoundedRectangle(74 * Width / 225, 0, 89 * Width / 225, Height);
00376             tbr[7] = new RoundedRectangle(89 * Width / 225, 0, 104 * Width / 225, Height);
00377             tbr[8] = new RoundedRectangle(104 * Width / 225, 0, 119 * Width / 225, Height);
00378             tbr[9] = new RoundedRectangle(119 * Width / 225, 0, 134 * Width / 225, Height);
00379             tbr[10] = new RoundedRectangle(134 * Width / 225, 0, 149 * Width / 225, Height);
00380             tbr[11] = new RoundedRectangle(149 * Width / 225, 0, 164 * Width / 225, Height);
00381             tbr[12] = new RoundedRectangle(164 * Width / 225, 0, 179 * Width / 225, Height);
00382             tbr[13] = new RoundedRectangle(179 * Width / 225, 0, 194 * Width / 225, Height);
00383             tbr[14] = new RoundedRectangle(194 * Width / 225, 0, 209 * Width / 225, Height);
00384             tbr[15] = new RoundedRectangle(209 * Width / 225, 0, Width, Height);
00385         }
00386         else if (divisions == 16)
00387         {
00388             tbr[0] = new RoundedRectangle(0, 0, Width / 16, Height);
00389             tbr[1] = new RoundedRectangle(Width / 16, 0, 15 * Width / 256, Height);
00390             tbr[2] = new RoundedRectangle(15 * Width / 256, 0, 31 * Width / 256, Height);
00391             tbr[3] = new RoundedRectangle(31 * Width / 256, 0, 47 * Width / 256, Height);
00392             tbr[4] = new RoundedRectangle(47 * Width / 256, 0, 63 * Width / 256, Height);
00393             tbr[5] = new RoundedRectangle(63 * Width / 256, 0, 79 * Width / 256, Height);
00394             tbr[6] = new RoundedRectangle(79 * Width / 256, 0, 95 * Width / 256, Height);
00395             tbr[7] = new RoundedRectangle(95 * Width / 256, 0, 111 * Width / 256, Height);
00396             tbr[8] = new RoundedRectangle(111 * Width / 256, 0, 127 * Width / 256, Height);
00397             tbr[9] = new RoundedRectangle(127 * Width / 256, 0, 143 * Width / 256, Height);
00398             tbr[10] = new RoundedRectangle(143 * Width / 256, 0, 159 * Width / 256, Height);
00399             tbr[11] = new RoundedRectangle(159 * Width / 256, 0, 175 * Width / 256, Height);
00400             tbr[12] = new RoundedRectangle(175 * Width / 256, 0, 191 * Width / 256, Height);
00401             tbr[13] = new RoundedRectangle(191 * Width / 256, 0, 207 * Width / 256, Height);
00402             tbr[14] = new RoundedRectangle(207 * Width / 256, 0, 223 * Width / 256, Height);
00403             tbr[15] = new RoundedRectangle(223 * Width / 256, 0, 239 * Width / 256, Height);
00404             tbr[16] = new RoundedRectangle(239 * Width / 256, 0, Width, Height);
00405         }
00406         else if (divisions == 17)
00407         {
00408             tbr[0] = new RoundedRectangle(0, 0, Width / 17, Height);
00409             tbr[1] = new RoundedRectangle(Width / 17, 0, 16 * Width / 289, Height);
00410             tbr[2] = new RoundedRectangle(16 * Width / 289, 0, 33 * Width / 289, Height);
00411             tbr[3] = new RoundedRectangle(33 * Width / 289, 0, 50 * Width / 289, Height);
00412             tbr[4] = new RoundedRectangle(50 * Width / 289, 0, 67 * Width / 289, Height);
00413             tbr[5] = new RoundedRectangle(67 * Width / 289, 0, 84 * Width / 289, Height);
00414             tbr[6] = new RoundedRectangle(84 * Width / 289, 0, 101 * Width / 289, Height);
00415             tbr[7] = new RoundedRectangle(101 * Width / 289, 0, 118 * Width / 289, Height);
00416             tbr[8] = new RoundedRectangle(118 * Width / 289, 0, 135 * Width / 289, Height);
00417             tbr[9] = new RoundedRectangle(135 * Width / 289, 0, 152 * Width / 289, Height);
00418             tbr[10] = new RoundedRectangle(152 * Width / 289, 0, 169 * Width / 289, Height);
00419             tbr[11] = new RoundedRectangle(169 * Width / 289, 0, 186 * Width / 289, Height);
00420             tbr[12] = new RoundedRectangle(186 * Width / 289, 0, 203 * Width / 289, Height);
00421             tbr[13] = new RoundedRectangle(203 * Width / 289, 0, 220 * Width / 289, Height);
00422             tbr[14] = new RoundedRectangle(220 * Width / 289, 0, 237 * Width / 289, Height);
00423             tbr[15] = new RoundedRectangle(237 * Width / 289, 0, 254 * Width / 289, Height);
00424             tbr[16] = new RoundedRectangle(254 * Width / 289, 0, 271 * Width / 289, Height);
00425             tbr[17] = new RoundedRectangle(271 * Width / 289, 0, Width, Height);
00426         }
00427         else if (divisions == 18)
00428         {
00429             tbr[0] = new RoundedRectangle(0, 0, Width / 18, Height);
00430             tbr[1] = new RoundedRectangle(Width / 18, 0, 17 * Width / 324, Height);
00431             tbr[2] = new RoundedRectangle(17 * Width / 324, 0, 35 * Width / 324, Height);
00432             tbr[3] = new RoundedRectangle(35 * Width / 324, 0, 52 * Width / 324, Height);
00433             tbr[4] = new RoundedRectangle(52 * Width / 324, 0, 69 * Width / 324, Height);
00434             tbr[5] = new RoundedRectangle(69 * Width / 324, 0, 86 * Width / 324, Height);
00435             tbr[6] = new RoundedRectangle(86 * Width / 324, 0, 103 * Width / 324, Height);
00436             tbr[7] = new RoundedRectangle(103 * Width / 324, 0, 120 * Width / 324, Height);
00437             tbr[8] = new RoundedRectangle(120 * Width / 324, 0, 137 * Width / 324, Height);
00438             tbr[9] = new RoundedRectangle(137 * Width / 324, 0, 154 * Width / 324, Height);
00439             tbr[10] = new RoundedRectangle(154 * Width / 324, 0, 171 * Width / 324, Height);
00440             tbr[11] = new RoundedRectangle(171 * Width / 324, 0, 188 * Width / 324, Height);
00441             tbr[12] = new RoundedRectangle(188 * Width / 324, 0, 205 * Width / 324, Height);
00442             tbr[13] = new RoundedRectangle(205 * Width / 324, 0, 222 * Width / 324, Height);
00443             tbr[14] = new RoundedRectangle(222 * Width / 324, 0, 239 * Width / 324, Height);
00444             tbr[15] = new RoundedRectangle(239 * Width / 324, 0, 256 * Width / 324, Height);
00445             tbr[16] = new RoundedRectangle(256 * Width / 324, 0, 273 * Width / 324, Height);
00446             tbr[17] = new RoundedRectangle(273 * Width / 324, 0, 290 * Width / 324, Height);
00447             tbr[18] = new RoundedRectangle(290 * Width / 324, 0, Width, Height);
00448         }
00449         else if (divisions == 19)
00450         {
00451             tbr[0] = new RoundedRectangle(0, 0, Width / 19, Height);
00452             tbr[1] = new RoundedRectangle(Width / 19, 0, 18 * Width / 361, Height);
00453             tbr[2] = new RoundedRectangle(18 * Width / 361, 0, 37 * Width / 361, Height);
00454             tbr[3] = new RoundedRectangle(37 * Width / 361, 0, 56 * Width / 361, Height);
00455             tbr[4] = new RoundedRectangle(56 * Width / 361, 0, 75 * Width / 361, Height);
00456             tbr[5] = new RoundedRectangle(75 * Width / 361, 0, 94 * Width / 361, Height);
00457             tbr[6] = new RoundedRectangle(94 * Width / 361, 0, 113 * Width / 361, Height);
00458             tbr[7] = new RoundedRectangle(113 * Width / 361, 0, 132 * Width / 361, Height);
00459             tbr[8] = new RoundedRectangle(132 * Width / 361, 0, 151 * Width / 361, Height);
00460             tbr[9] = new RoundedRectangle(151 * Width / 361, 0, 170 * Width / 361, Height);
00461             tbr[10] = new RoundedRectangle(170 * Width / 361, 0, 189 * Width / 361, Height);
00462             tbr[11] = new RoundedRectangle(189 * Width / 361, 0, 208 * Width / 361, Height);
00463             tbr[12] = new RoundedRectangle(208 * Width / 361, 0, 227 * Width / 361, Height);
00464             tbr[13] = new RoundedRectangle(227 * Width / 361, 0, 246 * Width / 361, Height);
00465             tbr[14] = new RoundedRectangle(246 * Width / 361, 0, 265 * Width / 361, Height);
00466             tbr[15] = new RoundedRectangle(265 * Width / 361, 0, 284 * Width / 361, Height);
00467             tbr[16] = new RoundedRectangle(284 * Width / 361, 0, 303 * Width / 361, Height);
00468             tbr[17] = new RoundedRectangle(303 * Width / 361, 0, 322 * Width / 361, Height);
00469             tbr[18] = new RoundedRectangle(322 * Width / 361, 0, 341 * Width / 361, Height);
00470             tbr[19] = new RoundedRectangle(341 * Width / 361, 0, Width, Height);
00471         }
00472         else if (divisions == 20)
00473         {
00474             tbr[0] = new RoundedRectangle(0, 0, Width / 20, Height);
00475             tbr[1] = new RoundedRectangle(Width / 20, 0, 19 * Width / 400, Height);
00476             tbr[2] = new RoundedRectangle(19 * Width / 400, 0, 39 * Width / 400, Height);
00477             tbr[3] = new RoundedRectangle(39 * Width / 400, 0, 58 * Width / 400, Height);
00478             tbr[4] = new RoundedRectangle(58 * Width / 400, 0, 77 * Width / 400, Height);
00479             tbr[5] = new RoundedRectangle(77 * Width / 400, 0, 96 * Width / 400, Height);
00480             tbr[6] = new RoundedRectangle(96 * Width / 400, 0, 115 * Width / 400, Height);
00481             tbr[7] = new RoundedRectangle(115 * Width / 400, 0, 134 * Width / 400, Height);
00482             tbr[8] = new RoundedRectangle(134 * Width / 400, 0, 153 * Width / 400, Height);
00483             tbr[9] = new RoundedRectangle(153 * Width / 400, 0, 172 * Width / 400, Height);
00484             tbr[10] = new RoundedRectangle(172 * Width / 400, 0, 191 * Width / 400, Height);
00485             tbr[11] = new RoundedRectangle(191 * Width / 400, 0, 210 * Width / 400, Height);
00486             tbr[12] = new RoundedRectangle(210 * Width / 400, 0, 229 * Width / 400, Height);
00487             tbr[13] = new RoundedRectangle(229 * Width / 400, 0, 248 * Width / 400, Height);
00488             tbr[14] = new RoundedRectangle(248 * Width / 400, 0, 267 * Width / 400, Height);
00489             tbr[15] = new RoundedRectangle(267 * Width / 400, 0, 286 * Width / 400, Height);
00490             tbr[16] = new RoundedRectangle(286 * Width / 400, 0, 305 * Width / 400, Height);
00491             tbr[17] = new RoundedRectangle(305 * Width / 400, 0, 324 * Width / 400, Height);
00492             tbr[18] = new RoundedRectangle(324 * Width / 400, 0, 343 * Width / 400, Height);
00493             tbr[19] = new RoundedRectangle(343 * Width / 400, 0, 362 * Width / 400, Height);
00494             tbr[20] = new RoundedRectangle(362 * Width / 400, 0, Width, Height);
00495         }
00496         else if (divisions == 21)
00497         {
00498             tbr[0] = new RoundedRectangle(0, 0, Width / 21, Height);
00499             tbr[1] = new RoundedRectangle(Width / 21, 0, 20 * Width / 441, Height);
00500             tbr[2] = new RoundedRectangle(20 * Width / 441, 0, 40 * Width / 441, Height);
00501             tbr[3] = new RoundedRectangle(40 * Width / 441, 0, 60 * Width / 441, Height);
00502             tbr[4] = new RoundedRectangle(60 * Width / 441, 0, 80 * Width / 441, Height);
00503             tbr[5] = new RoundedRectangle(80 * Width / 441, 0, 100 * Width / 441, Height);
00504             tbr[6] = new RoundedRectangle(100 * Width / 441, 0, 120 * Width / 441, Height);
00505             tbr[7] = new RoundedRectangle(120 * Width / 441, 0, 140 * Width / 441, Height);
00506             tbr[8] = new RoundedRectangle(140 * Width / 441, 0, 160 * Width / 441, Height);
00507             tbr[9] = new RoundedRectangle(160 * Width / 441, 0, 180 * Width / 441, Height);
00508             tbr[10] = new RoundedRectangle(180 * Width / 441, 0, 200 * Width / 441, Height);
00509             tbr[11] = new RoundedRectangle(200 * Width / 441, 0, 220 * Width / 441, Height);
00510             tbr[12] = new RoundedRectangle(220 * Width / 441, 0, 240 * Width / 441, Height);
00511             tbr[13] = new RoundedRectangle(240 * Width / 441, 0, 260 * Width / 441, Height);
00512             tbr[14] = new RoundedRectangle(260 * Width / 441
```

```

00222         {
00223             if (isVertical)
00224             {
00225                 tbr[0] = new RoundedRectangle(0, 0, Width, Height / 2);
00226                 tbr[1] = new RoundedRectangle(0, Height / 2, Width, Height);
00227             }
00228             else
00229             {
00230                 tbr[0] = new RoundedRectangle(0, 0, Width / 2, Height);
00231                 tbr[1] = new RoundedRectangle(Width / 2, 0, Width, Height);
00232             }
00233         }
00234         else if (divisions == 3)
00235         {
00236             if (isVertical)
00237             {
00238                 tbr[0] = new RoundedRectangle(0, 0, Width / 2, 2 * Height / 3);
00239                 tbr[1] = new RoundedRectangle(Width / 2, 0, Width, 2 * Height / 3);
00240                 tbr[2] = new RoundedRectangle(0, 2 * Height / 3, Width, Height);
00241             }
00242             else
00243             {
00244                 tbr[0] = new RoundedRectangle(0, 0, 2 * Width / 3, Height / 2);
00245                 tbr[1] = new RoundedRectangle(0, Height / 2, 2 * Width / 3, Height);
00246                 tbr[2] = new RoundedRectangle(2 * Width / 3, 0, Width, Height);
00247             }
00248         }
00249         else if (divisions == 5)
00250         {
00251             if (isVertical)
00252             {
00253                 tbr[0] = new RoundedRectangle(0, 0, Width / 2, 2 * Height / 5);
00254                 tbr[1] = new RoundedRectangle(Width / 2, 0, Width, 2 * Height / 5);
00255                 tbr[2] = new RoundedRectangle(0, 2 * Height / 5, Width / 2, 4 * Height / 5);
00256                 tbr[3] = new RoundedRectangle(Width / 2, 2 * Height / 5, Width, 4 * Height / 5);
00257                 tbr[4] = new RoundedRectangle(0, 4 * Height / 5, Width, Height);
00258             }
00259             else
00260             {
00261                 tbr[0] = new RoundedRectangle(0, 0, 2 * Width / 5, Height / 2);
00262                 tbr[1] = new RoundedRectangle(0, Height / 2, 2 * Width / 5, Height);
00263                 tbr[2] = new RoundedRectangle(2 * Width / 5, 0, 4 * Width / 5, Height / 2);
00264                 tbr[3] = new RoundedRectangle(2 * Width / 5, Height / 2, 4 * Width / 5, Height);
00265                 tbr[4] = new RoundedRectangle(4 * Width / 5, 0, Width, Height);
00266             }
00267         }
00268         else if (divisions == 7)
00269         {
00270             if (isVertical)
00271             {
00272                 tbr[0] = new RoundedRectangle(0, 0, Width / 2, 2 * Height / 7);
00273                 tbr[1] = new RoundedRectangle(Width / 2, 0, Width, 2 * Height / 7);
00274                 tbr[2] = new RoundedRectangle(0, 2 * Height / 7, Width / 2, 4 * Height / 7);
00275                 tbr[3] = new RoundedRectangle(Width / 2, 2 * Height / 7, Width, 4 * Height / 7);
00276                 tbr[4] = new RoundedRectangle(0, 4 * Height / 7, Width / 2, 6 * Height / 7);
00277                 tbr[5] = new RoundedRectangle(Width / 2, 4 * Height / 7, Width, 6 * Height / 7);
00278                 tbr[6] = new RoundedRectangle(0, 6 * Height / 7, Width, Height);
00279             }
00280             else
00281             {
00282                 tbr[0] = new RoundedRectangle(0, 0, 2 * Width / 7, Height / 2);
00283                 tbr[1] = new RoundedRectangle(0, Height / 2, 2 * Width / 7, Height);
00284                 tbr[2] = new RoundedRectangle(2 * Width / 7, 0, 4 * Width / 7, Height / 2);
00285                 tbr[3] = new RoundedRectangle(2 * Width / 7, Height / 2, 4 * Width / 7, Height);
00286                 tbr[4] = new RoundedRectangle(4 * Width / 7, 0, 6 * Width / 7, Height / 2);
00287                 tbr[5] = new RoundedRectangle(4 * Width / 7, Height / 2, 6 * Width / 7, Height);
00288                 tbr[6] = new RoundedRectangle(6 * Width / 7, 0, Width, Height);
00289             }
00290         }
00291         else
00292         {
00293             for (int divisorInd = 0; divisorInd < Utils.AcceptableDivisors.Length; divisorInd++)
00294             {
00295                 if (divisions % Utils.AcceptableDivisors[divisorInd] == 0)
00296                 {
00297                     RoundedRectangle[] largerDivisions = this.Split(divisions /
00298                         Utils.AcceptableDivisors[divisorInd]);
00299
00300                     int pos = 0;
00301
00302                     for (int i = 0; i < largerDivisions.Length; i++)
00303                     {
00304                         RoundedRectangle s = new RoundedRectangle(largerDivisions[i].Width,
00305                             largerDivisions[i].Height);
00306                         RoundedRectangle[] currDivision =
00307                             s.Split(Utils.AcceptableDivisors[divisorInd]);
00308
00309                 }
00310             }
00311         }
00312     }
00313 }
```

```
00306             for (int j = 0; j < currDivision.Length; j++)
00307             {
00308                 tbr[pos] = new RoundedRectangle(largerDivisions[i].X0 +
00309                     currDivision[j].X0, largerDivisions[i].Y0 + currDivision[j].Y0, largerDivisions[i].X0 +
00310                     currDivision[j].X1, largerDivisions[i].Y0 + currDivision[j].Y1);
00311                 pos++;
00312             }
00313         }
00314     }
00315 }
00316 }
00317
00318     return tbr;
00319 }
00320 }
00321 }
00322
00323 /// <summary>
00324 /// Represents a rectangle.
00325 /// </summary>
00326 public struct Rectangle
00327 {
00328     /// <summary>
00329     /// The left coordinate of the rectangle.
00330     /// </summary>
00331     public float X0;
00332
00333     /// <summary>
00334     /// The top coordinate of the rectangle.
00335     /// </summary>
00336     public float Y0;
00337
00338     /// <summary>
00339     /// The right coordinate of the rectangle.
00340     /// </summary>
00341     public float X1;
00342
00343     /// <summary>
00344     /// The bottom coordinate of the rectangle.
00345     /// </summary>
00346     public float Y1;
00347
00348     /// <summary>
00349     /// The width of the rectangle.
00350     /// </summary>
00351     public float Width => X1 - X0;
00352
00353     /// <summary>
00354     /// The height of the rectangle.
00355     /// </summary>
00356     public float Height => Y1 - Y0;
00357
00358     /// <summary>
00359     /// Create a new <see cref="Rectangle"/> from the specified coordinates.
00360     /// </summary>
00361     /// <param name="x0">The left coordinate of the rectangle.</param>
00362     /// <param name="y0">The top coordinate of the rectangle.</param>
00363     /// <param name="x1">The right coordinate of the rectangle.</param>
00364     /// <param name="y1">The bottom coordinate of the rectangle.</param>
00365     public Rectangle(float x0, float y0, float x1, float y1)
00366     {
00367         X0 = x0;
00368         Y0 = y0;
00369         X1 = x1;
00370         Y1 = y1;
00371     }
00372
00373     /// <summary>
00374     /// Create a new <see cref="Rectangle"/> from the specified coordinates.
00375     /// </summary>
00376     /// <param name="x0">The left coordinate of the rectangle.</param>
00377     /// <param name="y0">The top coordinate of the rectangle.</param>
00378     /// <param name="x1">The right coordinate of the rectangle.</param>
00379     /// <param name="y1">The bottom coordinate of the rectangle.</param>
00380     public Rectangle(double x0, double y0, double x1, double y1)
00381     {
00382         X0 = (float)x0;
00383         Y0 = (float)y0;
00384         X1 = (float)x1;
00385         Y1 = (float)y1;
00386     }
00387
00388     /// <summary>
00389     /// Round the rectangle's coordinates to the closest integers.
00390     /// </summary>
```

```

00391     /// <returns>A <see cref="RoundedRectangle"/> with the rounded coordinates.</returns>
00392     public RoundedRectangle Round()
00393     {
00394         return new RoundedRectangle(
00395             (int)Math.Floor(X0 + 0.001),
00396             (int)Math.Floor(Y0 + 0.001),
00397             (int)Math.Ceiling(X1 - 0.001),
00398             (int)Math.Ceiling(Y1 - 0.001)
00399         );
00400     }
00401
00402     /// <summary>
00403     /// Round the rectangle's coordinates to the closest integers, applying the specified zoom
00404     /// factor.
00405     /// </summary>
00406     /// <param name="zoom">The zoom factor to apply.</param>
00407     /// <returns>A <see cref="RoundedRectangle"/> with the rounded coordinates.</returns>
00408     public RoundedRectangle Round(double zoom)
00409     {
00410         return new RoundedRectangle(
00411             (int)Math.Floor(X0 * (float)zoom + 0.001),
00412             (int)Math.Floor(Y0 * (float)zoom + 0.001),
00413             (int)Math.Ceiling(X1 * (float)zoom - 0.001),
00414             (int)Math.Ceiling(Y1 * (float)zoom - 0.001)
00415         );
00416     }
00417     /// <summary>
00418     /// Split the rectangle into the specified number of <see cref="Rectangle"/>s.
00419     /// </summary>
00420     /// <param name="divisions">The number of rectangles in which the rectangle should be split.
00421     /// This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller
00422     /// than <paramref name="divisions"/> that satisfies this condition is used.</param>
00423     /// <returns>An array of <see cref="Rectangle"/>s that when positioned properly cover the same
00424     /// area as this object.</returns>
00425     public Rectangle[] Split(int divisions)
00426     {
00427         Size s = new Size(this.Width, this.Height);
00428
00429         Rectangle[] splitSize = s.Split(divisions);
00430
00431         Rectangle[] tbr = new Rectangle[divisions];
00432
00433         for (int i = 0; i < splitSize.Length; i++)
00434         {
00435             tbr[i] = new Rectangle(this.X0 + splitSize[i].X0, this.Y0 + splitSize[i].Y0, this.X0 +
00436             splitSize[i].X1, this.Y0 + splitSize[i].Y1);
00437         }
00438
00439         return tbr;
00440     }
00441
00442     /// <summary>
00443     /// Compute the intersection between this <see cref="Rectangle"/> and another one.
00444     /// </summary>
00445     /// <param name="other">The other <see cref="Rectangle"/> to intersect with this
00446     /// instance.</param>
00447     /// <returns>The intersection between the two <see cref="Rectangle"/>s.</returns>
00448     public Rectangle Intersect(Rectangle other)
00449     {
00450         float x0 = Math.Max(this.X0, other.X0);
00451         float y0 = Math.Max(this.Y0, other.Y0);
00452
00453         float x1 = Math.Min(this.X1, other.X1);
00454         float y1 = Math.Min(this.Y1, other.Y1);
00455
00456         if (x1 <= x0 || y1 <= y0)
00457         {
00458             return new Rectangle(0, 0, 0, 0);
00459         }
00460         else
00461         {
00462             return new Rectangle(x0, y0, x1, y1);
00463         }
00464
00465     /// <summary>
00466     /// Checks whether this <see cref="Rectangle"/> contains another <see cref="Rectangle"/>.
00467     /// </summary>
00468     /// <param name="other">The <see cref="Rectangle"/> to check.</param>
00469     /// <returns>A boolean value indicating whether this <see cref="Rectangle"/> contains the
00470     /// <paramref name="other"/> <see cref="Rectangle"/>.</returns>
00471     public bool Contains(Rectangle other)
00472     {
00473         return other.X0 >= this.X0 && other.X1 <= this.X1 && other.Y0 >= this.Y0 && other.Y1 <=
00474             this.Y1;
00475     }

```

```

00470     /// <summary>
00471     /// Checks whether this <see cref="Rectangle"/> contains a <see cref="PointF"/>.
00472     /// </summary>
00473     /// <param name="point">The <see cref="PointF"/> to check.</param>
00474     /// <returns>A boolean value indicating whether this <see cref="Rectangle"/> contains the
00475     /// <paramref name="point"/>.</returns>
00476     public bool Contains(PointF point)
00477     {
00478         return point.X >= this.X0 && point.X <= this.X1 && point.Y >= this.Y0 && point.Y <=
00479         this.Y1;
00480     }
00481
00482     /// <summary>
00483     /// Converts the <see cref="Rectangle"/> to a <see cref="Quad"/>.
00484     /// </summary>
00485     /// <returns>A <see cref="Quad"/> corresponding to the current <see
00486     /// cref="Rectangle"/>.</returns>
00487     public Quad ToQuad()
00488     {
00489         return new Quad(new PointF(X0, Y1), new PointF(X0, Y0), new PointF(X1, Y0), new PointF(X1,
00490         Y1));
00491     }
00492
00493     /// <summary>
00494     /// Represents a rectangle using only integer numbers.
00495     /// </summary>
00496     public struct RoundedRectangle
00497     {
00498         /// <summary>
00499         /// The left coordinate of the rectangle.
00500         /// </summary>
00501         public int X0;
00502
00503         /// <summary>
00504         /// The top coordinate of the rectangle.
00505         /// </summary>
00506         public int Y0;
00507
00508         /// <summary>
00509         /// The right coordinate of the rectangle.
00510         /// </summary>
00511         public int X1;
00512
00513         /// <summary>
00514         /// The bottom coordinate of the rectangle.
00515         /// </summary>
00516         public int Y1;
00517
00518         /// <summary>
00519         /// The width of the rectangle.
00520         /// </summary>
00521         public int Width => X1 - X0;
00522
00523         /// <summary>
00524         /// The height of the rectangle.
00525         /// </summary>
00526         public int Height => Y1 - Y0;
00527
00528         /// <summary>
00529         /// Create a new <see cref="RoundedRectangle"/> from the specified coordinates.
00530         /// </summary>
00531         /// <param name="x0">The left coordinate of the rectangle.</param>
00532         /// <param name="y0">The top coordinate of the rectangle.</param>
00533         /// <param name="x1">The right coordinate of the rectangle.</param>
00534         /// <param name="y1">The bottom coordinate of the rectangle.</param>
00535         public RoundedRectangle(int x0, int y0, int x1, int y1)
00536         {
00537             X0 = x0;
00538             Y0 = y0;
00539             X1 = x1;
00540             Y1 = y1;
00541
00542             /// <summary>
00543             /// Split the rectangle into the specified number of <see cref="RoundedRectangle"/>s.
00544             /// </summary>
00545             /// <param name="divisions">The number of rectangles in which the rectangle should be split.
00546             /// This must be factorisable using only powers of 2, 3, 5 or 7. Otherwise, the biggest number smaller
00547             /// than <paramref name="divisions"/> that satisfies this condition is used.</param>
00548             /// <returns>An array of <see cref="RoundedRectangle"/>s that when positioned properly cover
00549             /// the same area as this object.</returns>
00550             public RoundedRectangle[] Split(int divisions)
00551             {
00552                 RoundedSize s = new RoundedSize(this.Width, this.Height);
00553             }
00554         }
00555     }
00556 
```

```

00550         RoundedRectangle[] splitSize = s.Split(divisions);
00551
00552         RoundedRectangle[] tbr = new RoundedRectangle[divisions];
00553
00554         for (int i = 0; i < splitSize.Length; i++)
00555         {
00556             tbr[i] = new RoundedRectangle(this.X0 + splitSize[i].X0, this.Y0 + splitSize[i].Y0,
00557                                         this.X0 + splitSize[i].X1, this.Y0 + splitSize[i].Y1);
00558         }
00559
00560         return tbr;
00561     }
00562
00563     /// <summary>
00564     /// Represents a point.
00565     /// </summary>
00566     public struct PointF
00567     {
00568         /// <summary>
00569         /// The horizontal coordinate of the point.
00570         /// </summary>
00571         public float X;
00572
00573         /// <summary>
00574         /// The vertical coordinate of the point.
00575         /// </summary>
00576         public float Y;
00577
00578         /// <summary>
00579         /// Create a new <see cref="PointF"/> from the specified coordinates.
00580         /// </summary>
00581         /// <param name="x">The horizontal coordinate of the point.</param>
00582         /// <param name="y">The vertical coordinate of the point.</param>
00583         public PointF(float x, float y)
00584         {
00585             X = x;
00586             Y = y;
00587         }
00588     }
00589
00590     /// <summary>
00591     /// Represents a quadrilater (not necessarily a rectangle).
00592     /// </summary>
00593     public struct Quad
00594     {
00595         /// <summary>
00596         /// The lower left point of the quadrilater.
00597         /// </summary>
00598         public PointF LowerLeft;
00599
00600         /// <summary>
00601         /// The upper left point of the quadrilater.
00602         /// </summary>
00603         public PointF UpperLeft;
00604
00605         /// <summary>
00606         /// The upper right point of the quadrilater.
00607         /// </summary>
00608         public PointF UpperRight;
00609
00610         /// <summary>
00611         /// The lower right point of the quadrilater.
00612         /// </summary>
00613         public PointF LowerRight;
00614
00615         /// <summary>
00616         /// Creates a new <see cref="Quad"/> from the specified points.
00617         /// </summary>
00618         /// <param name="lowerLeft">The lower left point of the quadrilater.</param>
00619         /// <param name="upperLeft">The upper left point of the quadrilater.</param>
00620         /// <param name="upperRight">The upper right point of the quadrilater.</param>
00621         /// <param name="lowerRight">The lower right point of the quadrilater.</param>
00622         public Quad(PointF lowerLeft, PointF upperLeft, PointF upperRight, PointF lowerRight)
00623         {
00624             this.LowerLeft = lowerLeft;
00625             this.UpperLeft = upperLeft;
00626             this.UpperRight = upperRight;
00627             this.LowerRight = lowerRight;
00628         }
00629
00630         /// <summary>
00631         /// Checks whether this <see cref="Quad"/> contains a <see cref="PointF"/>.
00632         /// </summary>
00633         /// <param name="point">The <see cref="PointF"/> to check.</param>
00634         /// <returns>A boolean value indicating whether this <see cref="Quad"/> contains the <paramref
name="point"/>.</returns>

```

```

00635     public bool Contains(PointF point)
00636     {
00637         return PointInTriangle(point, this.LowerLeft, this.UpperLeft, this.UpperRight) ||
00638             PointInTriangle(point, this.LowerLeft, this.UpperRight, this.LowerRight);
00639     }
00640 
00641     /// <summary>
00642     /// Checks whether a point is contained in a triangle.
00643     /// </summary>
00644     /// <param name="pt">The point to test.</param>
00645     /// <param name="A">The first vertex of the triangle.</param>
00646     /// <param name="B">The second vertex of the triangle.</param>
00647     /// <param name="C">The third vertex of the triangle.</param>
00648     /// <returns>A boolean value indicating whether the point is contained in the
00649     /// triangle.</returns>
00650     private static bool PointInTriangle(PointF pt, PointF A, PointF B, PointF C)
00651     {
00652         double signAB = (pt.X - B.X) * (A.Y - B.Y) - (A.X - B.X) * (pt.Y - B.Y);
00653         double signBC = (pt.X - C.X) * (B.Y - C.Y) - (B.X - C.X) * (pt.Y - C.Y);
00654         double signCA = (pt.X - A.X) * (C.Y - A.Y) - (C.X - A.X) * (pt.Y - A.Y);
00655 
00656         return !((signAB < 0 || signBC < 0 || signCA < 0) && (signAB > 0 || signBC > 0 || signCA >
00657         0));
    }
}

```

8.12 TesseractLanguage.cs

```

00001 using System;
00002 using System.Collections.Generic;
00003 using System.IO;
00004 using System.Net;
00005 using System.Reflection;
00006 using System.Text;
00007
00008 namespace MuPDFCore
00009 {
00010     /// <summary>
00011     /// Represents a language used by Tesseract OCR.
00012     /// </summary>
00013     public class TesseractLanguage
00014     {
00015         /// <summary>
00016         /// The name of the folder where the language file is located.
00017         /// </summary>
00018         public string Prefix { get; }
00019 
00020         /// <summary>
00021         /// The name of the language. The Tesseract library will assume that the trained language data
00022         /// file can be found at <c>Prefix/Language.traineddata</c>.
00023         /// </summary>
00024         public string Language { get; }
00025 
00026         /// <summary>
00027         /// Fast integer versions of trained models. These are models for a single language.
00028         public enum Fast
00029         {
00030             /// <summary>
00031             /// The Afrikaans language.
00032             /// </summary>
00033             Afr,
00034             /// <summary>
00035             /// The Amharic language.
00036             /// </summary>
00037             Amh,
00038             /// <summary>
00039             /// The Arabic language.
00040             /// </summary>
00041             Ara,
00042             /// <summary>
00043             /// The Assamese language.
00044             /// </summary>
00045             Asm,
00046             /// <summary>
00047             /// The Azerbaijani language.
00048             /// </summary>
00049             Aze,
00050             /// <summary>
00051             /// The Azerbaijani language (Cyrillic).
00052             /// </summary>
00053             Aze_Cyrl,
        }
    }
}

```

```
00054     /// <summary>
00055     /// The Belarusian language.
00056     /// </summary>
00057     Bel,
00058     /// <summary>
00059     /// The Bengali language.
00060     /// </summary>
00061     Ben,
00062     /// <summary>
00063     /// The Tibetan language.
00064     /// </summary>
00065     Bod,
00066     /// <summary>
00067     /// The Bosnian language.
00068     /// </summary>
00069     Bos,
00070     /// <summary>
00071     /// The Breton language.
00072     /// </summary>
00073     Bre,
00074     /// <summary>
00075     /// The Bulgarian language.
00076     /// </summary>
00077     Bul,
00078     /// <summary>
00079     /// The Catalan/Valencian language.
00080     /// </summary>
00081     Cat,
00082     /// <summary>
00083     /// The Cebuano language.
00084     /// </summary>
00085     Ceb,
00086     /// <summary>
00087     /// The Czech language.
00088     /// </summary>
00089     Ces,
00090     /// <summary>
00091     /// The Chinese (Simplified) language.
00092     /// </summary>
00093     Chi_Sim,
00094     /// <summary>
00095     /// The Chinese (Simplified) language (vertical).
00096     /// </summary>
00097     Chi_Sim_Vert,
00098     /// <summary>
00099     /// The Chinese (Traditional) language.
00100     /// </summary>
00101     Chi_Tra,
00102     /// <summary>
00103     /// The Chinese (Traditional) language (vertical).
00104     /// </summary>
00105     Chi_Tra_Vert,
00106     /// <summary>
00107     /// The Cherokee language.
00108     /// </summary>
00109     Chr,
00110     /// <summary>
00111     /// The Corsican language.
00112     /// </summary>
00113     Cos,
00114     /// <summary>
00115     /// The Welsh language.
00116     /// </summary>
00117     Cym,
00118     /// <summary>
00119     /// The Danish language.
00120     /// </summary>
00121     Dan,
00122     /// <summary>
00123     /// The German language.
00124     /// </summary>
00125     Deu,
00126     /// <summary>
00127     /// The Divehi/Dhivehi/Maldivian language.
00128     /// </summary>
00129     Div,
00130     /// <summary>
00131     /// The Dzongkha language.
00132     /// </summary>
00133     Dzo,
00134     /// <summary>
00135     /// The Greek, Modern (1453-) language.
00136     /// </summary>
00137     Ell,
00138     /// <summary>
00139     /// The English language.
00140     /// </summary>
```

```
00141     Eng,
00142     /// <summary>
00143     /// The English, Middle (1100-1500) language.
00144     /// </summary>
00145     Enm,
00146     /// <summary>
00147     /// The Esperanto language.
00148     /// </summary>
00149     Epo,
00150     /// <summary>
00151     /// A language for equations.
00152     /// </summary>
00153     Equ,
00154     /// <summary>
00155     /// The Estonian language.
00156     /// </summary>
00157     Est,
00158     /// <summary>
00159     /// The Basque language.
00160     /// </summary>
00161     Eus,
00162     /// <summary>
00163     /// The Faroese language.
00164     /// </summary>
00165     Fao,
00166     /// <summary>
00167     /// The Persian language.
00168     /// </summary>
00169     Fas,
00170     /// <summary>
00171     /// The Filipino/Pilipino language.
00172     /// </summary>
00173     Fil,
00174     /// <summary>
00175     /// The Finnish language.
00176     /// </summary>
00177     Fin,
00178     /// <summary>
00179     /// The French language.
00180     /// </summary>
00181     Fra,
00182     /// <summary>
00183     /// The German - Fraktur language.
00184     /// </summary>
00185     Frk,
00186     /// <summary>
00187     /// The French, Middle (ca.1400-1600) language.
00188     /// </summary>
00189     Frm,
00190     /// <summary>
00191     /// The Western Frisian language.
00192     /// </summary>
00193     Fry,
00194     /// <summary>
00195     /// The Gaelic/Scottish Gaelic language.
00196     /// </summary>
00197     Gla,
00198     /// <summary>
00199     /// The Irish language.
00200     /// </summary>
00201     Gle,
00202     /// <summary>
00203     /// The Galician language.
00204     /// </summary>
00205     Glg,
00206     /// <summary>
00207     /// The Greek, Ancient (to 1453) language.
00208     /// </summary>
00209     Grc,
00210     /// <summary>
00211     /// The Gujarati language.
00212     /// </summary>
00213     Guj,
00214     /// <summary>
00215     /// The Haitian/Haitian Creole language.
00216     /// </summary>
00217     Hat,
00218     /// <summary>
00219     /// The Hebrew language.
00220     /// </summary>
00221     Heb,
00222     /// <summary>
00223     /// The Hindi language.
00224     /// </summary>
00225     Hin,
00226     /// <summary>
00227     /// The Croatian language.
```

```
00228      /// </summary>
00229      Hrv,
00230      /// <summary>
00231      /// The Hungarian language.
00232      /// </summary>
00233      Hun,
00234      /// <summary>
00235      /// The Armenian language.
00236      /// </summary>
00237      Hye,
00238      /// <summary>
00239      /// The Inuktitut language.
00240      /// </summary>
00241      Iku,
00242      /// <summary>
00243      /// The Indonesian language.
00244      /// </summary>
00245      Ind,
00246      /// <summary>
00247      /// The Icelandic language.
00248      /// </summary>
00249      Isl,
00250      /// <summary>
00251      /// The Italian language.
00252      /// </summary>
00253      Ita,
00254      /// <summary>
00255      /// The Italian language (old).
00256      /// </summary>
00257      Ita_Old,
00258      /// <summary>
00259      /// The Javanese language.
00260      /// </summary>
00261      Jav,
00262      /// <summary>
00263      /// The Japanese language.
00264      /// </summary>
00265      Jpn,
00266      /// <summary>
00267      /// The Japanese language (vertical).
00268      /// </summary>
00269      Jpn_Vert,
00270      /// <summary>
00271      /// The Kannada language.
00272      /// </summary>
00273      Kan,
00274      /// <summary>
00275      /// The Georgian language.
00276      /// </summary>
00277      Kat,
00278      /// <summary>
00279      /// The Georgian language (old).
00280      /// </summary>
00281      Kat_Old,
00282      /// <summary>
00283      /// The Kazakh language.
00284      /// </summary>
00285      Kaz,
00286      /// <summary>
00287      /// The Central Khmer language.
00288      /// </summary>
00289      Khm,
00290      /// <summary>
00291      /// The Kirghiz/Kyrgyz language.
00292      /// </summary>
00293      Kir,
00294      /// <summary>
00295      /// The Northern Kurdish language.
00296      /// </summary>
00297      Kmr,
00298      /// <summary>
00299      /// The Korean language.
00300      /// </summary>
00301      Kor,
00302      /// <summary>
00303      /// The Korean language (vertical).
00304      /// </summary>
00305      Kor_Vert,
00306      /// <summary>
00307      /// The Lao language.
00308      /// </summary>
00309      Lao,
00310      /// <summary>
00311      /// The Latin language.
00312      /// </summary>
00313      Lat,
00314      /// <summary>
```

```
00315     /// The Latvian language.  
00316     /// </summary>  
00317     Lav,  
00318     /// <summary>  
00319     /// The Lithuanian language.  
00320     /// </summary>  
00321     Lit,  
00322     /// <summary>  
00323     /// The Luxembourgish/Letzeburgesch language.  
00324     /// </summary>  
00325     Ltz,  
00326     /// <summary>  
00327     /// The Malayalam language.  
00328     /// </summary>  
00329     Mal,  
00330     /// <summary>  
00331     /// The Marathi language.  
00332     /// </summary>  
00333     Mar,  
00334     /// <summary>  
00335     /// The Macedonian language.  
00336     /// </summary>  
00337     Mkd,  
00338     /// <summary>  
00339     /// The Maltese language.  
00340     /// </summary>  
00341     Mlt,  
00342     /// <summary>  
00343     /// The Mongolian language.  
00344     /// </summary>  
00345     Mon,  
00346     /// <summary>  
00347     /// The Maori language.  
00348     /// </summary>  
00349     Mri,  
00350     /// <summary>  
00351     /// The Malay language.  
00352     /// </summary>  
00353     Msa,  
00354     /// <summary>  
00355     /// The Burmese language.  
00356     /// </summary>  
00357     Mya,  
00358     /// <summary>  
00359     /// The Nepali language.  
00360     /// </summary>  
00361     Nep,  
00362     /// <summary>  
00363     /// The Dutch/Flemish language.  
00364     /// </summary>  
00365     Nld,  
00366     /// <summary>  
00367     /// The Norwegian language.  
00368     /// </summary>  
00369     Nor,  
00370     /// <summary>  
00371     /// The Occitan (post 1500) language.  
00372     /// </summary>  
00373     Oci,  
00374     /// <summary>  
00375     /// The Oriya language.  
00376     /// </summary>  
00377     Ori,  
00378     /// <summary>  
00379     /// The Orientation and script detection module.  
00380     /// </summary>  
00381     Osd,  
00382     /// <summary>  
00383     /// The Panjabi/Punjabi language.  
00384     /// </summary>  
00385     Pan,  
00386     /// <summary>  
00387     /// The Polish language.  
00388     /// </summary>  
00389     Pol,  
00390     /// <summary>  
00391     /// The Portuguese language.  
00392     /// </summary>  
00393     Por,  
00394     /// <summary>  
00395     /// The Pushto/Pashto language.  
00396     /// </summary>  
00397     Pus,  
00398     /// <summary>  
00399     /// The Quechua language.  
00400     /// </summary>  
00401     Que,
```

```
00402      /// <summary>
00403      /// The Romanian/Moldavian/Moldovan language.
00404      /// </summary>
00405      Ron,
00406      /// <summary>
00407      /// The Russian language.
00408      /// </summary>
00409      Rus,
00410      /// <summary>
00411      /// The Sanskrit language.
00412      /// </summary>
00413      San,
00414      /// <summary>
00415      /// The Sinhala/Sinhalese language.
00416      /// </summary>
00417      Sin,
00418      /// <summary>
00419      /// The Slovak language.
00420      /// </summary>
00421      Slk,
00422      /// <summary>
00423      /// The Slovenian language.
00424      /// </summary>
00425      Slv,
00426      /// <summary>
00427      /// The Sindhi language.
00428      /// </summary>
00429      Snd,
00430      /// <summary>
00431      /// The Spanish/Castilian language.
00432      /// </summary>
00433      Spa,
00434      /// <summary>
00435      /// The Spanish/Castilian language (old).
00436      /// </summary>
00437      Spa_Old,
00438      /// <summary>
00439      /// The Albanian language.
00440      /// </summary>
00441      Sqi,
00442      /// <summary>
00443      /// The Serbian language.
00444      /// </summary>
00445      Srp,
00446      /// <summary>
00447      /// The Serbian language (Latin).
00448      /// </summary>
00449      Srp_Latn,
00450      /// <summary>
00451      /// The Sundanese language.
00452      /// </summary>
00453      Sun,
00454      /// <summary>
00455      /// The Swahili language.
00456      /// </summary>
00457      Swa,
00458      /// <summary>
00459      /// The Swedish language.
00460      /// </summary>
00461      Swe,
00462      /// <summary>
00463      /// The Syriac language.
00464      /// </summary>
00465      Syr,
00466      /// <summary>
00467      /// The Tamil language.
00468      /// </summary>
00469      Tam,
00470      /// <summary>
00471      /// The Tatar language.
00472      /// </summary>
00473      Tat,
00474      /// <summary>
00475      /// The Telugu language.
00476      /// </summary>
00477      Tel,
00478      /// <summary>
00479      /// The Tajik language.
00480      /// </summary>
00481      Tgk,
00482      /// <summary>
00483      /// The Thai language.
00484      /// </summary>
00485      Tha,
00486      /// <summary>
00487      /// The Tigrinya language.
00488      /// </summary>
```

```
00489     Tir,
00490     /// <summary>
00491     /// The Tonga (Tonga Islands) language.
00492     /// </summary>
00493     Ton,
00494     /// <summary>
00495     /// The Turkish language.
00496     /// </summary>
00497     Tur,
00498     /// <summary>
00499     /// The Uighur/Uyghur language.
00500     /// </summary>
00501     Uig,
00502     /// <summary>
00503     /// The Ukrainian language.
00504     /// </summary>
00505     Ukr,
00506     /// <summary>
00507     /// The Urdu language.
00508     /// </summary>
00509     Urd,
00510     /// <summary>
00511     /// The Uzbek language.
00512     /// </summary>
00513     Uzb,
00514     /// <summary>
00515     /// The Uzbek language (Cyrillic).
00516     /// </summary>
00517     Uzb_Cyrl,
00518     /// <summary>
00519     /// The Vietnamese language.
00520     /// </summary>
00521     Vie,
00522     /// <summary>
00523     /// The Yiddish language.
00524     /// </summary>
00525     Yid,
00526     /// <summary>
00527     /// The Yoruba language.
00528     /// </summary>
00529     Yor
00530 }
00531
00532     /// <summary>
00533     /// Fast integer versions of trained models. These are models for a single script supporting
00534     one or more languages.
00535     /// </summary>
00536     public enum FastScripts
00537     {
00538         /// <summary>
00539         /// The Arabic script.
00540         /// </summary>
00541         Arabic,
00542         /// <summary>
00543         /// The Armenian script.
00544         /// </summary>
00545         Armenian,
00546         /// <summary>
00547         /// The Bengali script.
00548         /// </summary>
00549         Bengali,
00550         /// <summary>
00551         /// The Canadian Aboriginal script.
00552         /// </summary>
00553         Canadian_Aboriginal,
00554         /// <summary>
00555         /// The Cherokee script.
00556         /// </summary>
00557         Cherokee,
00558         /// <summary>
00559         /// The Cyrillic script.
00560         /// </summary>
00561         Cyrillic,
00562         /// <summary>
00563         /// The Devanagari script.
00564         /// </summary>
00565         Devanagari,
00566         /// <summary>
00567         /// The Ethiopic script.
00568         /// </summary>
00569         Ethiopic,
00570         /// <summary>
00571         /// The Fraktur script.
00572         /// </summary>
00573         Fraktur,
00574         /// <summary>
00574         /// The Georgian script.
```

```
00575     /// </summary>
00576     Georgian,
00577     /// <summary>
00578     /// The Greek script.
00579     /// </summary>
00580     Greek,
00581     /// <summary>
00582     /// The Gujarati script.
00583     /// </summary>
00584     Gujarati,
00585     /// <summary>
00586     /// The Gurmukhi script.
00587     /// </summary>
00588     Gurmukhi,
00589     /// <summary>
00590     /// The Han (Simplified) script.
00591     /// </summary>
00592     HanS,
00593     /// <summary>
00594     /// The Han (Simplified) script. (vertical)
00595     /// </summary>
00596     Hans_Vert,
00597     /// <summary>
00598     /// The Han (Traditional) script.
00599     /// </summary>
00600     HanT,
00601     /// <summary>
00602     /// The Han (Traditional) script. (vertical)
00603     /// </summary>
00604     HanT_Vert,
00605     /// <summary>
00606     /// The Hangul script.
00607     /// </summary>
00608     Hangul,
00609     /// <summary>
00610     /// The Hangul script. (vertical)
00611     /// </summary>
00612     Hangul_Vert,
00613     /// <summary>
00614     /// The Hebrew script.
00615     /// </summary>
00616     Hebrew,
00617     /// <summary>
00618     /// The Japanese script.
00619     /// </summary>
00620     Japanese,
00621     /// <summary>
00622     /// The Japanese script. (vertical)
00623     /// </summary>
00624     Japanese_Vert,
00625     /// <summary>
00626     /// The Kannada script.
00627     /// </summary>
00628     Kannada,
00629     /// <summary>
00630     /// The Khmer script.
00631     /// </summary>
00632     Khmer,
00633     /// <summary>
00634     /// The Lao script.
00635     /// </summary>
00636     Lao,
00637     /// <summary>
00638     /// The Latin script.
00639     /// </summary>
00640     Latin,
00641     /// <summary>
00642     /// The Malayalam script.
00643     /// </summary>
00644     Malayalam,
00645     /// <summary>
00646     /// The Myanmar script.
00647     /// </summary>
00648     Myanmar,
00649     /// <summary>
00650     /// The Oriya script.
00651     /// </summary>
00652     Oriya,
00653     /// <summary>
00654     /// The Sinhala script.
00655     /// </summary>
00656     Sinhala,
00657     /// <summary>
00658     /// The Syriac script.
00659     /// </summary>
00660     Syriac,
00661     /// <summary>
```

```
00662     /// The Tamil script.  
00663     /// </summary>  
00664     Tamil,  
00665     /// <summary>  
00666     /// The Telugu script.  
00667     /// </summary>  
00668     Telugu,  
00669     /// <summary>  
00670     /// The Thaana script.  
00671     /// </summary>  
00672     Thaana,  
00673     /// <summary>  
00674     /// The Thai script.  
00675     /// </summary>  
00676     Thai,  
00677     /// <summary>  
00678     /// The Tibetan script.  
00679     /// </summary>  
00680     Tibetan,  
00681     /// <summary>  
00682     /// The Vietnamese script.  
00683     /// </summary>  
00684     Vietnamese  
00685 }  
00686  
00687     /// <summary>  
00688     /// Best (most accurate) trained models. These are models for a single language.  
00689     /// </summary>  
00690     public enum Best  
00691 {  
00692     /// <summary>  
00693     /// The Afrikaans language.  
00694     /// </summary>  
00695     Afr,  
00696     /// <summary>  
00697     /// The Amharic language.  
00698     /// </summary>  
00699     Amh,  
00700     /// <summary>  
00701     /// The Arabic language.  
00702     /// </summary>  
00703     Ara,  
00704     /// <summary>  
00705     /// The Assamese language.  
00706     /// </summary>  
00707     Asm,  
00708     /// <summary>  
00709     /// The Azerbaijani language.  
00710     /// </summary>  
00711     Aze,  
00712     /// <summary>  
00713     /// The Azerbaijani language (Cyrillic).  
00714     /// </summary>  
00715     Aze_Cyrl,  
00716     /// <summary>  
00717     /// The Belarusian language.  
00718     /// </summary>  
00719     Bel,  
00720     /// <summary>  
00721     /// The Bengali language.  
00722     /// </summary>  
00723     Ben,  
00724     /// <summary>  
00725     /// The Tibetan language.  
00726     /// </summary>  
00727     Bod,  
00728     /// <summary>  
00729     /// The Bosnian language.  
00730     /// </summary>  
00731     Bos,  
00732     /// <summary>  
00733     /// The Breton language.  
00734     /// </summary>  
00735     Bre,  
00736     /// <summary>  
00737     /// The Bulgarian language.  
00738     /// </summary>  
00739     Bul,  
00740     /// <summary>  
00741     /// The Catalan/Valencian language.  
00742     /// </summary>  
00743     Cat,  
00744     /// <summary>  
00745     /// The Cebuano language.  
00746     /// </summary>  
00747     Ceb,  
00748     /// <summary>
```

```
00749      /// The Czech language.  
00750      /// </summary>  
00751      Ces,  
00752      /// <summary>  
00753      /// The Chinese (Simplified) language.  
00754      /// </summary>  
00755      Chi_Sim,  
00756      /// <summary>  
00757      /// The Chinese (Simplified) language (vertical).  
00758      /// </summary>  
00759      Chi_Sim_Vert,  
00760      /// <summary>  
00761      /// The Chinese (Traditional) language.  
00762      /// </summary>  
00763      Chi_Tra,  
00764      /// <summary>  
00765      /// The Chinese (Traditional) language (vertical).  
00766      /// </summary>  
00767      Chi_Tra_Vert,  
00768      /// <summary>  
00769      /// The Cherokee language.  
00770      /// </summary>  
00771      Chr,  
00772      /// <summary>  
00773      /// The Corsican language.  
00774      /// </summary>  
00775      Cos,  
00776      /// <summary>  
00777      /// The Welsh language.  
00778      /// </summary>  
00779      Cym,  
00780      /// <summary>  
00781      /// The Danish language.  
00782      /// </summary>  
00783      Dan,  
00784      /// <summary>  
00785      /// The German language.  
00786      /// </summary>  
00787      Deu,  
00788      /// <summary>  
00789      /// The Divehi/Dhivehi/Maldivian language.  
00790      /// </summary>  
00791      Div,  
00792      /// <summary>  
00793      /// The Dzongkha language.  
00794      /// </summary>  
00795      Dzo,  
00796      /// <summary>  
00797      /// The Greek, Modern (1453-) language.  
00798      /// </summary>  
00799      Ell,  
00800      /// <summary>  
00801      /// The English language.  
00802      /// </summary>  
00803      Eng,  
00804      /// <summary>  
00805      /// The English, Middle (1100-1500) language.  
00806      /// </summary>  
00807      Enm,  
00808      /// <summary>  
00809      /// The Esperanto language.  
00810      /// </summary>  
00811      Epo,  
00812      /// <summary>  
00813      /// The Estonian language.  
00814      /// </summary>  
00815      Est,  
00816      /// <summary>  
00817      /// The Basque language.  
00818      /// </summary>  
00819      Eus,  
00820      /// <summary>  
00821      /// The Faroese language.  
00822      /// </summary>  
00823      Fao,  
00824      /// <summary>  
00825      /// The Persian language.  
00826      /// </summary>  
00827      Fas,  
00828      /// <summary>  
00829      /// The Filipino/Pilipino language.  
00830      /// </summary>  
00831      Fil,  
00832      /// <summary>  
00833      /// The Finnish language.  
00834      /// </summary>  
00835      Fin,
```

```
00836      /// <summary>
00837      /// The French language.
00838      /// </summary>
00839      Fra,
00840      /// <summary>
00841      /// The German - Fraktur language.
00842      /// </summary>
00843      Frk,
00844      /// <summary>
00845      /// The French, Middle (ca.1400-1600) language.
00846      /// </summary>
00847      Frm,
00848      /// <summary>
00849      /// The Western Frisian language.
00850      /// </summary>
00851      Fry,
00852      /// <summary>
00853      /// The Gaelic/Scottish Gaelic language.
00854      /// </summary>
00855      Gla,
00856      /// <summary>
00857      /// The Irish language.
00858      /// </summary>
00859      Gle,
00860      /// <summary>
00861      /// The Galician language.
00862      /// </summary>
00863      Glg,
00864      /// <summary>
00865      /// The Greek, Ancient (to 1453) language.
00866      /// </summary>
00867      Grc,
00868      /// <summary>
00869      /// The Gujarati language.
00870      /// </summary>
00871      Guj,
00872      /// <summary>
00873      /// The Haitian/Haitian Creole language.
00874      /// </summary>
00875      Hat,
00876      /// <summary>
00877      /// The Hebrew language.
00878      /// </summary>
00879      Heb,
00880      /// <summary>
00881      /// The Hindi language.
00882      /// </summary>
00883      Hin,
00884      /// <summary>
00885      /// The Croatian language.
00886      /// </summary>
00887      Hrv,
00888      /// <summary>
00889      /// The Hungarian language.
00890      /// </summary>
00891      Hun,
00892      /// <summary>
00893      /// The Armenian language.
00894      /// </summary>
00895      Hye,
00896      /// <summary>
00897      /// The Inuktitut language.
00898      /// </summary>
00899      Iku,
00900      /// <summary>
00901      /// The Indonesian language.
00902      /// </summary>
00903      Ind,
00904      /// <summary>
00905      /// The Icelandic language.
00906      /// </summary>
00907      Isl,
00908      /// <summary>
00909      /// The Italian language.
00910      /// </summary>
00911      Ita,
00912      /// <summary>
00913      /// The Italian language (old).
00914      /// </summary>
00915      Ita_Old,
00916      /// <summary>
00917      /// The Javanese language.
00918      /// </summary>
00919      Jav,
00920      /// <summary>
00921      /// The Japanese language.
00922      /// </summary>
```

```
00923     Jpn,
00924     /// <summary>
00925     /// The Japanese language (vertical).
00926     /// </summary>
00927     Jpn_Vert,
00928     /// <summary>
00929     /// The Kannada language.
00930     /// </summary>
00931     Kan,
00932     /// <summary>
00933     /// The Georgian language.
00934     /// </summary>
00935     Kat,
00936     /// <summary>
00937     /// The Georgian language (old).
00938     /// </summary>
00939     Kat_Old,
00940     /// <summary>
00941     /// The Kazakh language.
00942     /// </summary>
00943     Kaz,
00944     /// <summary>
00945     /// The Central Khmer language.
00946     /// </summary>
00947     Khm,
00948     /// <summary>
00949     /// The Kirghiz/Kyrgyz language.
00950     /// </summary>
00951     Kir,
00952     /// <summary>
00953     /// The Northern Kurdish language.
00954     /// </summary>
00955     Kmr,
00956     /// <summary>
00957     /// The Korean language.
00958     /// </summary>
00959     Kor,
00960     /// <summary>
00961     /// The Korean language (vertical).
00962     /// </summary>
00963     Kor_Vert,
00964     /// <summary>
00965     /// The Lao language.
00966     /// </summary>
00967     Lao,
00968     /// <summary>
00969     /// The Latin language.
00970     /// </summary>
00971     Lat,
00972     /// <summary>
00973     /// The Latvian language.
00974     /// </summary>
00975     Lav,
00976     /// <summary>
00977     /// The Lithuanian language.
00978     /// </summary>
00979     Lit,
00980     /// <summary>
00981     /// The Luxembourgish/Letzeburgesch language.
00982     /// </summary>
00983     Ltz,
00984     /// <summary>
00985     /// The Malayalam language.
00986     /// </summary>
00987     Mal,
00988     /// <summary>
00989     /// The Marathi language.
00990     /// </summary>
00991     Mar,
00992     /// <summary>
00993     /// The Macedonian language.
00994     /// </summary>
00995     Mkd,
00996     /// <summary>
00997     /// The Maltese language.
00998     /// </summary>
00999     Mlt,
01000     /// <summary>
01001     /// The Mongolian language.
01002     /// </summary>
01003     Mon,
01004     /// <summary>
01005     /// The Maori language.
01006     /// </summary>
01007     Mri,
01008     /// <summary>
01009     /// The Malay language.
```

```
01010      /// </summary>
01011      Msr,
01012      /// <summary>
01013      /// The Burmese language.
01014      /// </summary>
01015      Mya,
01016      /// <summary>
01017      /// The Nepali language.
01018      /// </summary>
01019      Nep,
01020      /// <summary>
01021      /// The Dutch/Flemish language.
01022      /// </summary>
01023      Nld,
01024      /// <summary>
01025      /// The Norwegian language.
01026      /// </summary>
01027      Nor,
01028      /// <summary>
01029      /// The Occitan (post 1500) language.
01030      /// </summary>
01031      Oci,
01032      /// <summary>
01033      /// The Oriya language.
01034      /// </summary>
01035      Ori,
01036      /// <summary>
01037      /// The Orientation and script detection module.
01038      /// </summary>
01039      Osd,
01040      /// <summary>
01041      /// The Panjabi/Punjabi language.
01042      /// </summary>
01043      Pan,
01044      /// <summary>
01045      /// The Polish language.
01046      /// </summary>
01047      Pol,
01048      /// <summary>
01049      /// The Portuguese language.
01050      /// </summary>
01051      Por,
01052      /// <summary>
01053      /// The Pushto/Pashto language.
01054      /// </summary>
01055      Pus,
01056      /// <summary>
01057      /// The Quechua language.
01058      /// </summary>
01059      Que,
01060      /// <summary>
01061      /// The Romanian/Moldavian/Moldovan language.
01062      /// </summary>
01063      Ron,
01064      /// <summary>
01065      /// The Russian language.
01066      /// </summary>
01067      Rus,
01068      /// <summary>
01069      /// The Sanskrit language.
01070      /// </summary>
01071      San,
01072      /// <summary>
01073      /// The Sinhala/Sinhalese language.
01074      /// </summary>
01075      Sin,
01076      /// <summary>
01077      /// The Slovak language.
01078      /// </summary>
01079      Slk,
01080      /// <summary>
01081      /// The Slovenian language.
01082      /// </summary>
01083      Slv,
01084      /// <summary>
01085      /// The Sindhi language.
01086      /// </summary>
01087      Snd,
01088      /// <summary>
01089      /// The Spanish/Castilian language.
01090      /// </summary>
01091      Spa,
01092      /// <summary>
01093      /// The Spanish/Castilian language (old).
01094      /// </summary>
01095      Spa_Old,
01096      /// <summary>
```

```
01097     /// The Albanian language.
01098     /// </summary>
01099     Sqi,
01100     /// <summary>
01101     /// The Serbian language.
01102     /// </summary>
01103     Srp,
01104     /// <summary>
01105     /// The Serbian language (Latin).
01106     /// </summary>
01107     Srp_Latn,
01108     /// <summary>
01109     /// The Sundanese language.
01110     /// </summary>
01111     Sun,
01112     /// <summary>
01113     /// The Swahili language.
01114     /// </summary>
01115     Swa,
01116     /// <summary>
01117     /// The Swedish language.
01118     /// </summary>
01119     Swe,
01120     /// <summary>
01121     /// The Syriac language.
01122     /// </summary>
01123     Syr,
01124     /// <summary>
01125     /// The Tamil language.
01126     /// </summary>
01127     Tam,
01128     /// <summary>
01129     /// The Tatar language.
01130     /// </summary>
01131     Tat,
01132     /// <summary>
01133     /// The Telugu language.
01134     /// </summary>
01135     Tel,
01136     /// <summary>
01137     /// The Tajik language.
01138     /// </summary>
01139     Tgk,
01140     /// <summary>
01141     /// The Thai language.
01142     /// </summary>
01143     Tha,
01144     /// <summary>
01145     /// The Tigrinya language.
01146     /// </summary>
01147     Tir,
01148     /// <summary>
01149     /// The Tonga (Tonga Islands) language.
01150     /// </summary>
01151     Ton,
01152     /// <summary>
01153     /// The Turkish language.
01154     /// </summary>
01155     Tur,
01156     /// <summary>
01157     /// The Uighur/Uyghur language.
01158     /// </summary>
01159     Uig,
01160     /// <summary>
01161     /// The Ukrainian language.
01162     /// </summary>
01163     Ukr,
01164     /// <summary>
01165     /// The Urdu language.
01166     /// </summary>
01167     Urd,
01168     /// <summary>
01169     /// The Uzbek language.
01170     /// </summary>
01171     Uzb,
01172     /// <summary>
01173     /// The Uzbek language (Cyrillic).
01174     /// </summary>
01175     Uzb_Cyril,
01176     /// <summary>
01177     /// The Vietnamese language.
01178     /// </summary>
01179     Vie,
01180     /// <summary>
01181     /// The Yiddish language.
01182     /// </summary>
01183     Yid,
```

```
01184     /// <summary>
01185     /// The Yoruba language.
01186     /// </summary>
01187     Yor
01188 }
01189
01190     /// <summary>
01191     /// Best (most accurate) trained models. These are models for a single script supporting one
01192     /// or more languages.
01193     /// </summary>
01194     public enum BestScripts
01195     {
01196         /// <summary>
01197         /// The Arabic script.
01198         /// </summary>
01199         Arabic,
01200         /// <summary>
01201         /// The Armenian script.
01202         /// </summary>
01203         Armenian,
01204         /// <summary>
01205         /// The Bengali script.
01206         /// </summary>
01207         Bengali,
01208         /// <summary>
01209         /// The Canadian Aboriginal script.
01210         /// </summary>
01211         Canadian_Aboriginal,
01212         /// <summary>
01213         /// The Cherokee script.
01214         /// </summary>
01215         Cherokee,
01216         /// <summary>
01217         /// The Cyrillic script.
01218         /// </summary>
01219         Cyrillic,
01220         /// <summary>
01221         /// The Devanagari script.
01222         /// </summary>
01223         Devanagari,
01224         /// <summary>
01225         /// The Ethiopic script.
01226         /// </summary>
01227         Ethiopic,
01228         /// <summary>
01229         /// The Fraktur script.
01230         /// </summary>
01231         Fraktur,
01232         /// <summary>
01233         /// The Georgian script.
01234         /// </summary>
01235         Georgian,
01236         /// <summary>
01237         /// The Greek script.
01238         /// </summary>
01239         Greek,
01240         /// <summary>
01241         /// The Gujarati script.
01242         /// </summary>
01243         Gujarati,
01244         /// <summary>
01245         /// The Gurmukhi script.
01246         /// </summary>
01247         Gurmukhi,
01248         /// <summary>
01249         /// The Han (Simplified) script.
01250         /// </summary>
01251         Hans,
01252         /// <summary>
01253         /// The Han (Simplified) script. (vertical)
01254         /// </summary>
01255         Hans_Vert,
01256         /// <summary>
01257         /// The Han (Traditional) script.
01258         /// </summary>
01259         HanT,
01260         /// <summary>
01261         /// The Han (Traditional) script. (vertical)
01262         /// </summary>
01263         HanT_Vert,
01264         /// <summary>
01265         /// The Hangul script.
01266         /// </summary>
01267         Hangul,
01268         /// <summary>
01269         /// The Hangul script. (vertical)
01270         /// </summary>
```

```

01270     Hangul_Vert,
01271     /// <summary>
01272     /// The Hebrew script.
01273     /// </summary>
01274     Hebrew,
01275     /// <summary>
01276     /// The Japanese script.
01277     /// </summary>
01278     Japanese,
01279     /// <summary>
01280     /// The Japanese script. (vertical)
01281     /// </summary>
01282     Japanese_Vert,
01283     /// <summary>
01284     /// The Kannada script.
01285     /// </summary>
01286     Kannada,
01287     /// <summary>
01288     /// The Khmer script.
01289     /// </summary>
01290     Khmer,
01291     /// <summary>
01292     /// The Lao script.
01293     /// </summary>
01294     Lao,
01295     /// <summary>
01296     /// The Latin script.
01297     /// </summary>
01298     Latin,
01299     /// <summary>
01300     /// The Malayalam script.
01301     /// </summary>
01302     Malayalam,
01303     /// <summary>
01304     /// The Myanmar script.
01305     /// </summary>
01306     Myanmar,
01307     /// <summary>
01308     /// The Oriya script.
01309     /// </summary>
01310     Oriya,
01311     /// <summary>
01312     /// The Sinhala script.
01313     /// </summary>
01314     Sinhala,
01315     /// <summary>
01316     /// The Syriac script.
01317     /// </summary>
01318     Syriac,
01319     /// <summary>
01320     /// The Tamil script.
01321     /// </summary>
01322     Tamil,
01323     /// <summary>
01324     /// The Telugu script.
01325     /// </summary>
01326     Telugu,
01327     /// <summary>
01328     /// The Thaana script.
01329     /// </summary>
01330     Thaana,
01331     /// <summary>
01332     /// The Thai script.
01333     /// </summary>
01334     Thai,
01335     /// <summary>
01336     /// The Tibetan script.
01337     /// </summary>
01338     Tibetan,
01339     /// <summary>
01340     /// The Vietnamese script.
01341     /// </summary>
01342     Vietnamese
01343 }
01344
01345     /// <summary>
01346     /// Create a new <see cref="TesseractLanguage"/> object using the provided <paramref
01347     name="prefix"/> and <paramref name="language"/> name, without processing them in any way.
01348     /// </summary>
01349     /// <param name="prefix">The name of the folder where the language file is located. If this is
01350     <see langword="null" />, the value of the environment variable <c>TESSDATA_PREFIX</c> will be
01351     used.</param>
01352     /// <param name="language">The name of the language. The Tesseract library will assume that
01353     the trained language data file can be found at <paramref name="prefix"/><c>/<c><paramref
01354     name="language"/><c>.traineddata</c></param>
01355     public TesseractLanguage(string prefix, string language)
01356     {

```

```

01352         this.Prefix = prefix;
01353         this.Language = language;
01354     }
01355
01356     /// <summary>
01357     /// Create a new <see cref="TesseractLanguage"/> object using the specified trained model data
01358     /// file.
01359     /// </summary>
01360     /// <param name="fileName">The path to the trained model data file. If the file name does not
01361     end in <c>.traineddata</c>, the file is copied to a temporary folder, and the temporary file is used
01362     by the Tesseract library.</param>
01363     public TesseractLanguage(string fileName)
01364     {
01365         if (fileName.EndsWith(".traineddata"))
01366         {
01367             fileName = Path.GetFullPath(fileName);
01368
01369             this.Prefix = Path.GetDirectoryName(fileName);
01370             this.Language = Path.GetFileName(fileName).Substring(0,
01371                 Path.GetFileName(fileName).Length - 12);
01372
01373             this.Prefix = Path.GetTempPath();
01374             this.Language = Guid.NewGuid().ToString("N");
01375
01376             File.Copy(fileName, Path.Combine(this.Prefix, this.Language + ".traineddata"));
01377         }
01378
01379         private static readonly string ExecutablePath =
01380             Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
01381         private static readonly string LocalCachePath =
01382             Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
01383
01384         /// <summary>
01385         /// Create a new <see cref="TesseractLanguage"/> object using a fast integer version of a
01386         trained model for the specified language. The language file is downloaded from the
01387         <c>tesseract-ocr/tessdata_fast</c> GitHub repository. If it has already been downloaded and cached
01388         before, the downloaded file is re-used.
01389         /// </summary>
01390         /// <param name="language">The language to use for the OCR process.</param>
01391         /// <param name="useAnyCached">If this is <see langword="true"/>, if a cached trained model
01392         file is available for the specified language, it will be used even if it is a "best (most accurate)"  

01393         model. Otherwise, only cached fast integer trained models will be used.</param>
01394         public TesseractLanguage(Fast language, bool useAnyCached = false)
01395         {
01396             string languageName = language.ToString().ToLower();
01397             string prefix = null;
01398
01399             if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01400                 "tessdata", "fast", languageName + ".traineddata")))
01401             {
01402                 prefix = Path.Combine(ExecutablePath, "tessdata", "fast");
01403             }
01404             else if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01405                 "fast", languageName + ".traineddata")))
01406             {
01407                 prefix = Path.Combine(ExecutablePath, "fast");
01408             }
01409             else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "fast", languageName +
01410                 ".traineddata")))
01411             {
01412                 prefix = Path.Combine(LocalCachePath, "tessdata", "fast");
01413             }
01414             else if (useAnyCached)
01415             {
01416                 if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01417                     languageName + ".traineddata")))
01418                 {
01419                     prefix = Path.Combine(ExecutablePath);
01420                 }
01421                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
01422                     File.Exists(Path.Combine(ExecutablePath, "tessdata", "best", languageName + ".traineddata")))
01423                 {
01424                     prefix = Path.Combine(ExecutablePath, "tessdata", "best");
01425                 }
01426                 else if (!string.IsNullOrEmpty(ExecutablePath) &&
01427                     File.Exists(Path.Combine(ExecutablePath, "best", languageName + ".traineddata")))
01428                 {
01429                     prefix = Path.Combine(ExecutablePath, "best");
01430                 }
01431                 else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "best", languageName +
01432                     ".traineddata")))
01433                 {
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02510
02511
02512
02513
02514
02515
02516
02517
02518
02519
02520
02521
02522
02523
02524
02525
02526
02527
02528
02529
02530
02531
02532
02533
02534
02535
02536
02537
02538
02539
02540
02541
02542
02543
02544
02545
02546
02547
02548
02549
02550
02551
02552
02553
02554
02555
02556
02557
02558
02559
02560
02561
02562
02563
02564
02565
02566
02567
02568
02569
02570
02571
02572
02573
02574
02575
02576
02577
02578
02579
02580
02581
02582
02583
02584
02585
02586
02587
02588
02589
02590
02591
02592
02593
02594
02595
02596
02597
02598
02599
02600
02601
02602
02603
02604
02605
02606
02607
02608
02609
02610
02611
02612
02613
02614
02615
02616
02617
02618
02619
02620
02621
02622
02623
02624
02625
02626
02627
02628
02629
02630
02631
02632
02633
02634
02635
02636
02637
02638
02639
02640
02641
02642
02643
02644
02645
02646
02647
02648
02649
02650
02651
02652
02653
02654
02655
02656
02657
02658
02659
02660
02661
02662
02663
02664
02665
02666
02667
02668
02669
02670
02671
02672
02673
02674
02675
02676
02677
02678
02679
02680
02681
02682
02683
02684
02685
02686
02687
02688
02689
02690
02691
02692
02693
02694
02695
02696
02697
02698
02699
02700
02701
02702
02703
02704
02705
02706
02707
02708
02709
02710
02711
02712
02713
02714
02715
02716
02717
02718
02719
02720
02721
02722
02723
02724
02725
02726
02727
02728
02729
02730
02731
02732
02733
02734
02735
02736
02737
02738
02739
02740
02741
02742
02743
02744
02745
02746
02747
02748
02749
02750
02751
02752
02753
02754
02755
02756
02757
02758
02759
02760
02761
02762
02763
02764
02765
02766
02767
02768
02769
02770
02771
02772
02773
02774
02775
02776
02777
02778
02779
02780
02781
02782
02783
02784
02785
02786
02787
02788
02789
02790
02791
02792
02793
02794
02795
02796
02797
02798
02799
02800
02801
02802
02803
02804
02805
02806
02807
02808
02809
02810
02811
02812
02813
02814
02815
02816
02817
02818
02819
02820
02821
02822
02823
02824
02825
02826
02827
02828
02829
02830
02831
02832
02833
02834
02835
02836
02837
02838
02839
02840
02841
02842
02843
02844
02845
02846
02847
02848
02849
02850
02851
02852
02853
02854
02855
02856
02857
02858
02859
02860
02861
02862
02863
02864
02865
02866
02867
02868
02869
02870
02871
02872
02873
02874
02875
02876
02877
02878
02879
02880
02881
02882
02883
02884
02885
02886
02887
02888
02889
02890
02891
02892
02893
02894
02895
02896
02897
02898
02899
02900
02901
02902
02903
02904
02905
02906
02907
02908
02909
02910
02911
02912
02913
02914
02915
02916
02917
02918
02919
02920
02921
02922
02923
02924
02925
02926
02927
02928
02929
02930
02931
02932
02933
02934
02935
02936
02937
02938
02939
02940
02941
02942
02943
02944
02945
02946
02947
02948
02949
02950
02951
02952
02953
02954
02955
02956
02957
02958
02959
02960
02961
02962
02963
02964
02965
02966
02967
02968
02969
02970
02971
02972
02973
02974
02975
02976
02977
02978
02979
02980
02981
02982
02983
02984
02985
02986
02987
02988
02989
02990
02991
02992
02993
02994
02995
02996
02997
02998
02999
03000
03001
03002
03003
03004
03005
03006
03007
03008
03009
03010
03011
03012
03013
03014
03015
03016
03017
03018
03019
03020
03021
03022
03023
03024
03025
03026
03027
03028
03029
03030
03031
03032
03033
03034
03035
03036
03037
03038
03039
03040
03041
03042
03043
03044
03045
03046
03047
03048
03049
03050
03051
03052
03053
03054
03055
03056
03057
03058
03059
03060
03061
03062
03063
03064
03065
03066
03067
```

```

01421             prefix = Path.Combine(LocalCachePath, "tessdata", "best");
01422         }
01423     }
01424 
01425     if (prefix == null)
01426     {
01427         string remotePath = "https://github.com/tesseract-ocr/tessdata_fast/raw/main/" +
languageName + ".traineddata";
01428 
01429         string localDirectory = Path.Combine(LocalCachePath, "tessdata", "fast");
01430 
01431         if (!Directory.Exists(localDirectory))
01432         {
01433             Directory.CreateDirectory(localDirectory);
01434         }
01435 
01436         using (WebClient client = new WebClient())
01437         {
01438             client.DownloadFile(remotePath, Path.Combine(localDirectory, languageName +
".traineddata"));
01439         }
01440 
01441         prefix = localDirectory;
01442     }
01443 
01444     this.Prefix = prefix;
01445     this.Language = languageName;
01446 }
01447 
01448 /// <summary>
01449 /// Create a new <see cref="TesseractLanguage"/> object using the best (most accurate) version
of the trained model for the specified language. The language file is downloaded from the
<c>tesseract-ocr/tessdata_best</c> GitHub repository. If it has already been downloaded and cached
before, the downloaded file is re-used.
01450 /// </summary>
01451 /// <param name="language">The language to use for the OCR process.</param>
01452 /// <param name="useAnyCached">If this is <see langword="true"/>, if a cached trained model
file is available for the specified language, it will be used even if it is a "fast" model.
Otherwise, only cached best (most accurate) trained models will be used.</param>
01453 public TesseractLanguage(Best language, bool useAnyCached = false)
01454 {
01455     string languageName = language.ToString().ToLower();
01456 
01457     string prefix = null;
01458 
01459     if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
"tessdata", "best", languageName + ".traineddata")))
01460     {
01461         prefix = Path.Combine(ExecutablePath, "tessdata", "best");
01462     }
01463     else if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
"best", languageName + ".traineddata")))
01464     {
01465         prefix = Path.Combine(ExecutablePath, "best");
01466     }
01467     else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "best", languageName +
".traineddata")))
01468     {
01469         prefix = Path.Combine(LocalCachePath, "tessdata", "best");
01470     }
01471     else if (useAnyCached)
01472     {
01473         if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
languageName + ".traineddata")))
01474         {
01475             prefix = Path.Combine(ExecutablePath);
01476         }
01477         else if (!string.IsNullOrEmpty(ExecutablePath) &&
File.Exists(Path.Combine(ExecutablePath, "tessdata", "fast", languageName + ".traineddata")))
01478         {
01479             prefix = Path.Combine(ExecutablePath, "tessdata", "fast");
01480         }
01481         else if (!string.IsNullOrEmpty(ExecutablePath) &&
File.Exists(Path.Combine(ExecutablePath, "fast", languageName + ".traineddata")))
01482         {
01483             prefix = Path.Combine(ExecutablePath, "fast");
01484         }
01485         else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "fast", languageName +
".traineddata")))
01486         {
01487             prefix = Path.Combine(LocalCachePath, "tessdata", "fast");
01488         }
01489     }
01490 
01491     if (prefix == null)
01492     {
01493         string remotePath = "https://github.com/tesseract-ocr/tessdata_best/raw/main/" +

```

```

        languageName + ".traineddata";
01494     string localDirectory = Path.Combine(LocalCachePath, "tessdata", "best");
01495     if (!Directory.Exists(localDirectory))
01496     {
01497         Directory.CreateDirectory(localDirectory);
01498     }
01499     using (WebClient client = new WebClient())
01500     {
01501         client.DownloadFile(remotePath, Path.Combine(localDirectory, languageName +
01502             ".traineddata"));
01503     }
01504     prefix = localDirectory;
01505 }
01506
01507     this.Prefix = prefix;
01508     this.Language = languageName;
01509 }
01510
01511 /// <summary>
01512 /// Create a new <see cref="TesseractLanguage"/> object using a fast integer version of a
01513 /// trained model for the specified script. The language file is downloaded from the
01514 /// <c>tesseract-ocr/tessdata_fast</c> GitHub repository. If it has already been downloaded and cached
01515 /// before, the downloaded file is re-used.
01516 /// </summary>
01517 /// <param name="script">The script to use for the OCR process.</param>
01518 /// <param name="useAnyCached">If this is <see langword="true"/>, if a cached trained model
01519 /// file is available for the specified script, it will be used even if it is a "best (most accurate)"  

01520 /// model. Otherwise, only cached fast integer trained models will be used.</param>
01521     public TesseractLanguage(FastScripts script, bool useAnyCached = false)
01522     {
01523         string languageName = script.ToString().Replace("_Vert", "_vert");
01524
01525         if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01526             "tessdata", "fast", "script", languageName + ".traineddata")))
01527         {
01528             prefix = Path.Combine(ExecutablePath, "tessdata", "fast", "script");
01529         }
01530         else if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01531             "fast", "script", languageName + ".traineddata")))
01532         {
01533             prefix = Path.Combine(ExecutablePath, "fast", "script");
01534         }
01535         else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "fast", "script",
01536             languageName + ".traineddata")))
01537         {
01538             prefix = Path.Combine(LocalCachePath, "tessdata", "fast", "script");
01539         }
01540         else if (useAnyCached)
01541         {
01542             if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01543                 "script", languageName + ".traineddata")))
01544             {
01545                 prefix = Path.Combine(ExecutablePath, "script");
01546             }
01547             else if (!string.IsNullOrEmpty(ExecutablePath) &&
01548                 File.Exists(Path.Combine(ExecutablePath, languageName + ".traineddata")))
01549             {
01550                 prefix = Path.Combine(ExecutablePath);
01551             }
01552             else if (!string.IsNullOrEmpty(ExecutablePath) &&
01553                 File.Exists(Path.Combine(ExecutablePath, "tessdata", "best", "script",
01554                     languageName + ".traineddata")))
01555             {
01556                 prefix = Path.Combine(ExecutablePath, "tessdata", "best", "script");
01557             }
01558             else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "best", "script",
01559             languageName + ".traineddata")))
01560             {
01561                 prefix = Path.Combine(LocalCachePath, "tessdata", "best", "script");
01562             }
01563         }
01564         if (prefix == null)
01565         {
01566             string remotePath = "https://github.com/tesseract-ocr/tessdata_fast/raw/main/script/" +
01567             languageName + ".traineddata";
01568         }
01569     }
01570 }
```

```

01564         string localDirectory = Path.Combine(LocalCachePath, "tessdata", "fast", "script");
01565
01566         if (!Directory.Exists(localDirectory))
01567         {
01568             Directory.CreateDirectory(localDirectory);
01569         }
01570
01571         using (WebClient client = new WebClient())
01572         {
01573             client.DownloadFile(remotePath, Path.Combine(localDirectory, languageName +
01574 ".traineddata"));
01575         }
01576
01577         prefix = localDirectory;
01578     }
01579
01580     this.Prefix = prefix;
01581     this.Language = languageName;
01582 }
01583
01584 /// <summary>
01585     /// Create a new <see cref="TesseractLanguage"/> object using the best (most accurate) version
01586     /// of the trained model for the specified script. The language file is downloaded from the
01587     /// <c>tesseract-ocr/tessdata_best</c> GitHub repository. If it has already been downloaded and cached
01588     /// before, the downloaded file is re-used.
01589     /// </summary>
01590     /// <param name="script">The script to use for the OCR process.</param>
01591     /// <param name="useAnyCached">If this is <see langword="true"/>, if a cached trained model
01592     /// file is available for the specified script, it will be used even if it is a "fast" model. Otherwise,
01593     /// only cached best (most accurate) trained models will be used.</param>
01594     public TesseractLanguage(BestScripts script, bool useAnyCached = false)
01595     {
01596         string languageName = script.ToString().Replace("_Vert", "_vert");
01597
01598         string prefix = null;
01599
01600         if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01601 "tessdata", "best", "script", languageName + ".traineddata")))
01602         {
01603             prefix = Path.Combine(ExecutablePath, "tessdata", "best", "script");
01604         }
01605         else if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01606 "best", "script", languageName + ".traineddata")))
01607         {
01608             prefix = Path.Combine(ExecutablePath, "best", "script");
01609         }
01610         else if (File.Exists(Path.Combine(LocalCachePath, "tessdata", "best", "script",
01611 languageName + ".traineddata")))
01612         {
01613             prefix = Path.Combine(LocalCachePath, "tessdata", "best", "script");
01614         }
01615         else if (useAnyCached)
01616         {
01617             if (!string.IsNullOrEmpty(ExecutablePath) && File.Exists(Path.Combine(ExecutablePath,
01618 "script", languageName + ".traineddata")))
01619             {
01620                 prefix = Path.Combine(ExecutablePath, "script");
01621             }
01622             else if (!string.IsNullOrEmpty(ExecutablePath) &&
01623 File.Exists(Path.Combine(ExecutablePath, languageName + ".traineddata")))
01624             {
01625                 prefix = Path.Combine(ExecutablePath);
01626             }
01627             else if (!string.IsNullOrEmpty(ExecutablePath) &&
01628 File.Exists(Path.Combine(ExecutablePath, "tessdata", "fast", "script", languageName +
01629 ".traineddata")))
01630             {
01631                 prefix = Path.Combine(ExecutablePath, "tessdata", "fast", "script");
01632             }
01633             if (prefix == null)
01634             {
01635                 string remotePath = "https://github.com/tesseract-ocr/tessdata_best/raw/main/script/" +
languageName + ".traineddata";
01636             }
01637         }
01638     }
01639 
```

```

01635         string localDirectory = Path.Combine(LocalCachePath, "tessdata", "best", "script");
01636
01637         if (!Directory.Exists(localDirectory))
01638         {
01639             Directory.CreateDirectory(localDirectory);
01640         }
01641
01642         using (WebClient client = new WebClient())
01643         {
01644             client.DownloadFile(remotePath, Path.Combine(localDirectory, languageName +
01645                         ".traineddata"));
01646         }
01647
01648         prefix = localDirectory;
01649     }
01650
01651         this.Prefix = prefix;
01652         this.Language = languageName;
01653     }
01654 }
```

8.13 Utils.cs

```

00001 /*
00002  MuPDFCore - A set of multiplatform .NET Core bindings for MuPDF.
00003  Copyright (C) 2020 Giorgio Bianchini
00004
00005  This program is free software: you can redistribute it and/or modify
00006  it under the terms of the GNU Affero General Public License as
00007  published by the Free Software Foundation, version 3.
00008
00009  This program is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU Affero General Public License for more details.
00013
00014  You should have received a copy of the GNU Affero General Public License
00015  along with this program. If not, see <http://www.gnu.org/licenses/>
00016 */
00017
00018 using System;
00019
00020 namespace MuPDFCore
00021 {
00022     /// <summary>
00023     /// Contains useful methods.
00024     /// </summary>
00025     internal static class Utils
00026     {
00027         /// <summary>
00028         /// The factors by which we can divide a rectangle/size.
00029         /// </summary>
00030         public static readonly int[] AcceptableDivisors = new[] { 2, 3, 5, 7 };
00031
00032         /// <summary>
00033         /// Computes the biggest number smaller than or equal to the specified value that is
00034         /// factorisable using only the <see cref="AcceptableDivisors"/>.
00035         /// </summary>
00036         /// <param name="n">The maximum number.</param>
00037         /// <returns>A number that is &lt;= <paramref name="n"/> and is factorisable using only the
00038         /// <see cref="AcceptableDivisors"/>.</returns>
00039         public static int GetAcceptableNumber(int n)
00040         {
00041             for (int i = n; i >= 1; i--)
00042             {
00043                 if (IsAcceptableNumber(i))
00044                 {
00045                     return i;
00046                 }
00047             }
00048             throw new ArgumentOutOfRangeException(nameof(n), n, "The number must be strictly higher
than 0!");
00049         }
00050
00051         /// <summary>
00052         /// Determine whether a number is factorisable using only the <see
00053         /// <param name="n">The number to analyse.</param>
00054         /// <returns>A boolean value indicating whether the number is factorisable using only the <see
00055         /// <see cref="AcceptableDivisors"/>.</returns>
```

```
00055     public static bool IsAcceptableNumber(int n)
00056     {
00057         if (n == 0)
00058         {
00059             return false;
00060         }
00061         else if (n == 1)
00062         {
00063             return true;
00064         }
00065         else
00066         {
00067             for (int i = 0; i < AcceptableDivisors.Length; i++)
00068             {
00069                 bool divided = false;
00070
00071                 while (n % AcceptableDivisors[i] == 0)
00072                 {
00073                     n /= AcceptableDivisors[i];
00074                     divided = true;
00075                 }
00076
00077                 if (divided)
00078                 {
00079                     return IsAcceptableNumber(n);
00080                 }
00081             }
00082
00083             return false;
00084         }
00085     }
00086
00087     /// <summary>
00088     /// Clear all pixels outside of a specified region.
00089     /// </summary>
00090     /// <param name="image">A pointer to the address where the pixel data is stored.</param>
00091     /// <param name="imageSize">The size in pixels of the image.</param>
00092     /// <param name="imageArea">The area represented by the image.</param>
00093     /// <param name="clipArea">The region outside which all pixels will be cleared, in image
units.<param>
00094     /// <param name="pixelFormat">The format of the pixel data.</param>
00095     public static void ClipImage(IntPtr image, RoundedSize imageSize, Rectangle imageArea,
Rectangle clipArea, PixelFormats pixelFormat)
00096     {
00097         int clipLeft = Math.Max(0, (int)Math.Ceiling((clipArea.X0 - imageArea.X0) /
imageArea.Width * imageSize.Width - 0.001));
00098         int clipRight = Math.Max(0, (int)Math.Floor(imageSize.Width - (imageArea.X1 - clipArea.X1) /
imageArea.Width * imageSize.Width + 0.001));
00099
00100         int clipTop = Math.Max(0, (int)Math.Ceiling((clipArea.Y0 - imageArea.Y0) /
imageArea.Height * imageSize.Height - 0.001));
00101         int clipBottom = Math.Max(0, (int)Math.Floor(imageSize.Height - (imageArea.Y1 -
clipArea.Y1) / imageArea.Height * imageSize.Height + 0.001));
00102
00103         int pixelSize = -1;
00104         byte clearValue = 0;
00105
00106         switch (pixelFormat)
00107         {
00108             case PixelFormats.RGB:
00109             case PixelFormats.BGR:
00110                 pixelSize = 3;
00111                 clearValue = 255;
00112                 break;
00113             case PixelFormats.RGBA:
00114             case PixelFormats.BGRA:
00115                 pixelSize = 4;
00116                 clearValue = 0;
00117                 break;
00118         }
00119
00120         int stride = imageSize.Width * pixelSize;
00121
00122         if (clipLeft > 0 || clipRight < imageSize.Width || clipTop > 0 || clipBottom <
imageSize.Height)
00123         {
00124             unsafe
00125             {
00126                 byte* imageData = (byte*)image;
00127
00128                 for (int y = 0; y < imageSize.Height; y++)
00129                 {
00130                     if (y < clipTop || y >= clipBottom)
00131                     {
00132                         for (int x = 0; x < imageSize.Width; x++)
00133                         {
00134                             for (int i = 0; i < pixelSize; i++)
00135                             {
00136                                 if (i < clipLeft || i >= clipRight)
00137                                 {
00138                                     imageData[y * stride + x * pixelSize + i] = clearValue;
00139                                 }
00140                             }
00141                         }
00142                     }
00143                 }
00144             }
00145         }
00146     }
00147 }
```

```
00135             {
00136                 imageData[y * stride + x * pixelSize + i] = clearValue;
00137             }
00138         }
00139     }
00140     else
00141     {
00142         for (int x = 0; x < Math.Min(clipLeft, imageSize.Width); x++)
00143         {
00144             for (int i = 0; i < pixelSize; i++)
00145             {
00146                 imageData[y * stride + x * pixelSize + i] = clearValue;
00147             }
00148         }
00149
00150         for (int x = Math.Max(0, clipRight); x < imageSize.Width; x++)
00151         {
00152             for (int i = 0; i < pixelSize; i++)
00153             {
00154                 imageData[y * stride + x * pixelSize + i] = clearValue;
00155             }
00156         }
00157     }
00158 }
00159 }
00160 }
00161 }
00162
00163 /// <summary>
00164 /// Converts an image with premultiplied alpha values into an image with unpremultiplied alpha
00165 /// values.
00166 /// </summary>
00167 /// <param name="image">A pointer to the address where the pixel data is stored.</param>
00168 /// <param name="imageSize">The size in pixels of the image.</param>
00169 public static void UnpremultiplyAlpha(IntPtr image, RoundedSize imageSize)
00170 {
00171     int stride = imageSize.Width * 4;
00172
00173     unsafe
00174     {
00175         byte* imageData = (byte*)image;
00176
00177         for (int y = 0; y < imageSize.Height; y++)
00178         {
00179             for (int x = 0; x < imageSize.Width; x++)
00180             {
00181                 if (imageData[y * stride + x * 4 + 3] > 0)
00182                 {
00183                     imageData[y * stride + x * 4] = (byte)(imageData[y * stride + x * 4] * 255
00184 / imageData[y * stride + x * 4 + 3]);
00185                     imageData[y * stride + x * 4 + 1] = (byte)(imageData[y * stride + x * 4 +
00186 1] * 255 / imageData[y * stride + x * 4 + 3]);
00187                     imageData[y * stride + x * 4 + 2] = (byte)(imageData[y * stride + x * 4 +
00188 2] * 255 / imageData[y * stride + x * 4 + 3]);
00189                 }
00190             }
00191         }
00192     }
00193 }
```


Index

Abort
 MuPDFCore.MuPDFMultiThreadedPageRenderer, [62](#)

Afr
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Amh
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Annotate
 MuPDFCore, [26](#)

Ara
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Arabic
 MuPDFCore.TesseractLanguage, [136](#), [140](#)

Armenian
 MuPDFCore.TesseractLanguage, [136](#), [141](#)

Asm
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Avalonia, [23](#)

Avalonia.Animation, [23](#)

Avalonia.Animation.RectTransition, [122](#)
 DoTransition, [122](#)

Aze
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Aze_Cyril
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Background
 MuPDFCore.MuPDFRenderer.PDFRenderer, [109](#)

BackgroundProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, [105](#)

Bel
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Ben
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Bengali
 MuPDFCore.TesseractLanguage, [137](#), [141](#)

Best
 MuPDFCore.TesseractLanguage, [133](#)

BestScripts
 MuPDFCore.TesseractLanguage, [136](#)

BGR
 MuPDFCore, [28](#)

BGRA
 MuPDFCore, [28](#)

BlockIndex
 MuPDFCore.MuPDFStructuredTextAddress, [74](#)

BMP
 MuPDFCore, [27](#)

Bod
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Bos
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

BoundingBox
 MuPDFCore.MuPDFStructuredTextBlock, [78](#)
 MuPDFCore.MuPDFStructuredTextLine, [83](#)

BoundingQuad
 MuPDFCore.MuPDFStructuredTextCharacter, [80](#)

Bounds
 MuPDFCore.MuPDFPage, [65](#)

Bre
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Bul
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Canadian_Aboriginal
 MuPDFCore.TesseractLanguage, [137](#), [141](#)

Cat
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

CBZ
 MuPDFCore, [25](#), [27](#)

Ceb
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Ces
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Character
 MuPDFCore.MuPDFStructuredTextCharacter, [80](#)

CharacterIndex
 MuPDFCore.MuPDFStructuredTextAddress, [74](#)

Characters
 MuPDFCore.MuPDFStructuredTextLine, [83](#)

Cherokee
 MuPDFCore.TesseractLanguage, [137](#), [141](#)

Chi_Sim
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Chi_Sim_Vert
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Chi_Tra
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Chi_Tra_Vert
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

Chr
 MuPDFCore.TesseractLanguage, [134](#), [138](#)

ClearCache
 MuPDFCore.MuPDFDocument, [44](#)

ClearStore
 MuPDFCore.MuPDFContext, [37](#)

ClipToPageBounds
 MuPDFCore.MuPDFDocument, [57](#)

CodePoint
 MuPDFCore.MuPDFStructuredTextCharacter, [80](#)

Color
 MuPDFCore.MuPDFStructuredTextCharacter, 80

CompareTo
 MuPDFCore.MuPDFStructuredTextAddress, 70

Contain
 MuPDFCore.MuPDFRenderer.PDFRenderer, 97

Contains
 MuPDFCore.Quad, 114
 MuPDFCore.Rectangle, 118

Copy
 MuPDFCore, 26

Cos
 MuPDFCore.TesseractLanguage, 134, 138

Count
 MuPDFCore.MuPDFImageStructuredTextBlock, 60
 MuPDFCore.MuPDFPageCollection, 67
 MuPDFCore.MuPDFStructuredTextBlock, 78
 MuPDFCore.MuPDFStructuredTextLine, 83
 MuPDFCore.MuPDFStructuredTextPage, 88
 MuPDFCore.MuPDFTextStructuredTextBlock, 91

Cover
 MuPDFCore.MuPDFRenderer.PDFRenderer, 97

CreateDocument
 MuPDFCore.MuPDFDocument, 44

Custom
 MuPDFCore.MuPDFRenderer.PDFRenderer, 96

Cym
 MuPDFCore.TesseractLanguage, 134, 138

Cyrillic
 MuPDFCore.TesseractLanguage, 137, 141

Dan
 MuPDFCore.TesseractLanguage, 134, 138

Deu
 MuPDFCore.TesseractLanguage, 134, 138

Devanagari
 MuPDFCore.TesseractLanguage, 137, 141

Direction
 MuPDFCore.MuPDFStructuredTextLine, 84

DisplayArea
 MuPDFCore.MuPDFRenderer.PDFRenderer, 109

DisplayAreaProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 105

DisposableIntPtr
 MuPDFCore.DisposableIntPtr, 32

Dispose
 MuPDFCore.DisposableIntPtr, 32
 MuPDFCore.MuPDFContext, 38
 MuPDFCore.MuPDFDocument, 46
 MuPDFCore.MuPDFMultiThreadedPageRenderer, 62
 MuPDFCore.MuPDFPage, 65
 MuPDFCore.MuPDFPageCollection, 67

Div
 MuPDFCore.TesseractLanguage, 134, 138

DocumentOutputFileTypes
 MuPDFCore, 25

DocumentRestrictions
 MuPDFCore, 25

DoTransition
 Avalonia.Animation.RectTransition, 122

Dzo
 MuPDFCore.TesseractLanguage, 134, 138

Edit
 MuPDFCore, 26

Ell
 MuPDFCore.TesseractLanguage, 134, 138

Encrypted
 MuPDFCore, 26

EncryptionState
 MuPDFCore, 26
 MuPDFCore.MuPDFDocument, 57

End
 MuPDFCore.MuPDFStructuredTextAddressSpan, 75

Eng
 MuPDFCore.TesseractLanguage, 134, 138

Enm
 MuPDFCore.TesseractLanguage, 134, 138

Epo
 MuPDFCore.TesseractLanguage, 134, 138

EPUB
 MuPDFCore, 27

Equ
 MuPDFCore.TesseractLanguage, 138

Equals
 MuPDFCore.MuPDFStructuredTextAddress, 70

ERR_CANNOT_CLONE_CONTEXT
 MuPDFCore, 26

ERR_CANNOT_CLOSE_DOCUMENT
 MuPDFCore, 27

ERR_CANNOT_COMPUTE_BOUNDS
 MuPDFCore, 26

ERR_CANNOT_COUNT_PAGES
 MuPDFCore, 26

ERR_CANNOT_CREATE_BUFFER
 MuPDFCore, 27

ERR_CANNOT_CREATE_CONTEXT
 MuPDFCore, 26

ERR_CANNOT_CREATE_PAGE
 MuPDFCore, 27

ERR_CANNOT_CREATE_WRITER
 MuPDFCore, 27

ERR_CANNOT_INIT_MUTEX
 MuPDFCore, 26

ERR_CANNOT_LOAD_PAGE
 MuPDFCore, 26

ERR_CANNOT_OPEN_FILE
 MuPDFCore, 26

ERR_CANNOT_OPEN_STREAM
 MuPDFCore, 26

ERR_CANNOT_POPULATE_PAGE
 MuPDFCore, 27

ERR_CANNOT_REGISTER_HANDLERS
 MuPDFCore, 26

ERR_CANNOT_RENDER
 MuPDFCore, 26

ERR_CANNOT_SAVE
 MuPDFCore, 27

ErrorCode
 MuPDFCore.MuPDFException, 59

Est
 MuPDFCore.TesseractLanguage, 134, 138

Ethiopic
 MuPDFCore.TesseractLanguage, 137, 141

Eus
 MuPDFCore.TesseractLanguage, 134, 138

EXIT_SUCCESS
 MuPDFCore, 27

ExitCodes
 MuPDFCore, 26

ExtractText
 MuPDFCore.MuPDFDocument, 46

ExtractTextAsync
 MuPDFCore.MuPDFDocument, 47

Fao
 MuPDFCore.TesseractLanguage, 134, 138

Fas
 MuPDFCore.TesseractLanguage, 134, 138

Fast
 MuPDFCore.TesseractLanguage, 137

FastScripts
 MuPDFCore.TesseractLanguage, 140

FB2
 MuPDFCore, 27

Fil
 MuPDFCore.TesseractLanguage, 134, 138

Fin
 MuPDFCore.TesseractLanguage, 134, 138

Fra
 MuPDFCore.TesseractLanguage, 134, 138

Fraktur
 MuPDFCore.TesseractLanguage, 137, 141

Frk
 MuPDFCore.TesseractLanguage, 134, 138

Frm
 MuPDFCore.TesseractLanguage, 134, 138

Fry
 MuPDFCore.TesseractLanguage, 134, 138

Georgian
 MuPDFCore.TesseractLanguage, 137, 141

GetClosestHitAddress
 MuPDFCore.MuPDFStructuredTextPage, 86

GetEnumerator
 MuPDFCore.MuPDFImageStructuredTextBlock, 60
 MuPDFCore.MuPDFPageCollection, 67
 MuPDFCore.MuPDFStructuredTextBlock, 77
 MuPDFCore.MuPDFStructuredTextLine, 82
 MuPDFCore.MuPDFStructuredTextPage, 86
 MuPDFCore.MuPDFTextStructuredTextBlock, 91

GetHashCode
 MuPDFCore.MuPDFStructuredTextAddress, 71

GetHighlightQuads
 MuPDFCore.MuPDFStructuredTextPage, 86

GetHitAddress
 MuPDFCore.MuPDFStructuredTextPage, 87

GetMultiThreadedRenderer
 MuPDFCore.MuPDFDocument, 48

GetProgress
 MuPDFCore.MuPDFMultiThreadedPageRenderer,
 62
 MuPDFCore.MuPDFRenderer.PDFRenderer, 97

GetRenderedSize
 MuPDFCore.MuPDFDocument, 48, 49

GetSelectedText
 MuPDFCore.MuPDFRenderer.PDFRenderer, 97

GetSpanItem
 MuPDFCore.MuPDFMultiThreadedPageRenderer,
 62

GetStructuredTextPage
 MuPDFCore.MuPDFDocument, 49, 50

GetStructuredTextPageAsync
 MuPDFCore.MuPDFDocument, 50

GetText
 MuPDFCore.MuPDFStructuredTextPage, 87

GIF
 MuPDFCore, 27

Gla
 MuPDFCore.TesseractLanguage, 134, 138

Gle
 MuPDFCore.TesseractLanguage, 134, 138

Glg
 MuPDFCore.TesseractLanguage, 134, 138

Grc
 MuPDFCore.TesseractLanguage, 134, 138

Greek
 MuPDFCore.TesseractLanguage, 137, 141

Guj
 MuPDFCore.TesseractLanguage, 134, 138

Gujarati
 MuPDFCore.TesseractLanguage, 137, 141

Gurmukhi
 MuPDFCore.TesseractLanguage, 137, 141

Hangul
 MuPDFCore.TesseractLanguage, 137, 141

Hangul_Vert
 MuPDFCore.TesseractLanguage, 137, 141

HanS
 MuPDFCore.TesseractLanguage, 137, 141

HanS_Vert
 MuPDFCore.TesseractLanguage, 137, 141

HanT
 MuPDFCore.TesseractLanguage, 137, 141

HanT_Vert
 MuPDFCore.TesseractLanguage, 137, 141

Hat
 MuPDFCore.TesseractLanguage, 134, 139

Heb
 MuPDFCore.TesseractLanguage, 135, 139

Hebrew
 MuPDFCore.TesseractLanguage, 137, 141

Height

MuPDFCore.Rectangle, 121
 MuPDFCore.RoundedRectangle, 126
 MuPDFCore.RoundedSize, 128
 MuPDFCore.Size, 130
Highlight
 MuPDFCore.MuPDFRenderer.PDFRenderer, 96
HighlightBrush
 MuPDFCore.MuPDFRenderer.PDFRenderer, 109
HighlightBrushProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 105
HighlightedRegions
 MuPDFCore.MuPDFRenderer.PDFRenderer, 109
HighlightedRegionsProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 106
Hin
 MuPDFCore.TesseractLanguage, 135, 139
Horizontal
 MuPDFCore.MuPDFStructuredTextLine, 82
Hrv
 MuPDFCore.TesseractLanguage, 135, 139
Hun
 MuPDFCore.TesseractLanguage, 135, 139
Hye
 MuPDFCore.TesseractLanguage, 135, 139
Iku
 MuPDFCore.TesseractLanguage, 135, 139
Image
 MuPDFCore.MuPDFStructuredTextBlock, 77
Increment
 MuPDFCore.MuPDFStructuredTextAddress, 71
Ind
 MuPDFCore.TesseractLanguage, 135, 139
Initialize
 MuPDFCore.MuPDFRenderer.PDFRenderer, 98–
 100
InitializeAsync
 MuPDFCore.MuPDFRenderer.PDFRenderer, 100–
 103
InputFileTypes
 MuPDFCore, 27
Intersect
 MuPDFCore.Rectangle, 119
Isl
 MuPDFCore.TesseractLanguage, 135, 139
IsViewerInitialized
 MuPDFCore.MuPDFRenderer.PDFRenderer, 110
IsViewerInitializedProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 106
Ita
 MuPDFCore.TesseractLanguage, 135, 139
Ita_Old
 MuPDFCore.TesseractLanguage, 135, 139
Japanese
 MuPDFCore.TesseractLanguage, 137, 141
Japanese_Vert
 MuPDFCore.TesseractLanguage, 137, 141
Jav

MuPDFCore.TesseractLanguage, 135, 139
JPEG
 MuPDFCore, 27
Jpn
 MuPDFCore.TesseractLanguage, 135, 139
Jpn_Vert
 MuPDFCore.TesseractLanguage, 135, 139
Kan
 MuPDFCore.TesseractLanguage, 135, 139
Kannada
 MuPDFCore.TesseractLanguage, 137, 141
Kat
 MuPDFCore.TesseractLanguage, 135, 139
Kat_Old
 MuPDFCore.TesseractLanguage, 135, 139
Kaz
 MuPDFCore.TesseractLanguage, 135, 139
Khm
 MuPDFCore.TesseractLanguage, 135, 139
Khmer
 MuPDFCore.TesseractLanguage, 137, 141
Kir
 MuPDFCore.TesseractLanguage, 135, 139
Kmr
 MuPDFCore.TesseractLanguage, 135, 139
Kor
 MuPDFCore.TesseractLanguage, 135, 139
Kor_Vert
 MuPDFCore.TesseractLanguage, 135, 139
Language
 MuPDFCore.TesseractLanguage, 144
Lao
 MuPDFCore.TesseractLanguage, 135, 137, 139,
 141
Lat
 MuPDFCore.TesseractLanguage, 135, 139
Latin
 MuPDFCore.TesseractLanguage, 137, 141
Lav
 MuPDFCore.TesseractLanguage, 135, 139
Length
 MuPDFCore.MuPDFPageCollection, 67
LineIndex
 MuPDFCore.MuPDFStructuredTextAddress, 74
Lines
 MuPDFCore.MuPDFTextStructuredTextBlock, 91
Lit
 MuPDFCore.TesseractLanguage, 135, 139
LowerLeft
 MuPDFCore.Quad, 115
LowerRight
 MuPDFCore.Quad, 115
Ltz
 MuPDFCore.TesseractLanguage, 135, 139
Mal
 MuPDFCore.TesseractLanguage, 135, 139

Malayalam
 MuPDFCore.TesseractLanguage, 137, 141

Mar
 MuPDFCore.TesseractLanguage, 135, 139

MaxProgress
 MuPDFCore.RenderProgress.ThreadRenderProgress, 145

Message
 MuPDFCore.MessageEventArgs, 34

MessageEventArgs
 MuPDFCore.MessageEventArgs, 34

Mkd
 MuPDFCore.TesseractLanguage, 135, 139

Mlt
 MuPDFCore.TesseractLanguage, 135, 139

Mon
 MuPDFCore.TesseractLanguage, 135, 139

Mri
 MuPDFCore.TesseractLanguage, 135, 139

Msa
 MuPDFCore.TesseractLanguage, 135, 139

MuPDFContext
 MuPDFCore.MuPDFContext, 37

MuPDFCore, 23
 Annotate, 26
 BGR, 28
 BGRA, 28
 BMP, 27
 CBZ, 25, 27
 Copy, 26
 DocumentOutputFileTypes, 25
 DocumentRestrictions, 25
 Edit, 26
 Encrypted, 26
 EncryptionState, 26
 EPUB, 27
 ERR_CANNOT_CLONE_CONTEXT, 26
 ERR_CANNOT_CLOSE_DOCUMENT, 27
 ERR_CANNOT_COMPUTE_BOUNDS, 26
 ERR_CANNOT_COUNT_PAGES, 26
 ERR_CANNOT_CREATE_BUFFER, 27
 ERR_CANNOT_CREATE_CONTEXT, 26
 ERR_CANNOT_CREATE_PAGE, 27
 ERR_CANNOT_CREATE_WRITER, 27
 ERR_CANNOT_INIT_MUTEX, 26
 ERR_CANNOT_LOAD_PAGE, 26
 ERR_CANNOT_OPEN_FILE, 26
 ERR_CANNOT_OPEN_STREAM, 26
 ERR_CANNOT_POPULATE_PAGE, 27
 ERR_CANNOT_REGISTER_HANDLERS, 26
 ERR_CANNOT_RENDER, 26
 ERR_CANNOT_SAVE, 27
 EXIT_SUCCESS, 27
 ExitCodes, 26
 FB2, 27
 GIF, 27
 InputFileTypes, 27
 JPEG, 27

None, 26, 27
Owner, 27
PAM, 27, 28
PasswordTypes, 27
PDF, 25, 27
PixelFormats, 28
PNG, 27, 28
PNM, 27, 28
Print, 26
PSD, 28
RasterOutputFileTypes, 28
Restricted, 28
RestrictionState, 28
RGB, 28
RGBA, 28
SVG, 25
TIFF, 27
Unencrypted, 26
Unlocked, 26, 28
Unrestricted, 28
User, 27
XPS, 27

MuPDFCore.DisposableIntPtr, 31
DisposableIntPtr, 32
Dispose, 32

MuPDFCore.DocumentLockedException, 33

MuPDFCore.MessageEventArgs, 33
 Message, 34
 MessageEventArgs, 34

MuPDFCore.MuPDF, 35
 RedirectOutput, 35
 ResetOutput, 35
 StandardErrorMessage, 36
 StandardOutputMessage, 36

MuPDFCore.MuPDFContext, 36
 ClearStore, 37
 Dispose, 38
 MuPDFContext, 37
 ShrinkStore, 38
 StoreMaxSize, 38
 StoreSize, 38

MuPDFCore.MuPDFDocument, 39
 ClearCache, 44
 ClipToPageBounds, 57
 CreateDocument, 44
 Dispose, 46
 EncryptionState, 57
 ExtractText, 46
 ExtractTextAsync, 47
 GetMultiThreadedRenderer, 48
 GetRenderedSize, 48, 49
 GetStructuredTextPage, 49, 50
 GetStructuredTextPageAsync, 50
 MuPDFDocument, 41–43
 Pages, 57
 Render, 51–54
 Restrictions, 58
 RestrictionState, 58

SaveImage, 54, 55
 TryUnlock, 55, 56
 WriteImage, 56, 57
MuPDFCore.MuPDFException, 58
 ErrorCode, 59
MuPDFCore.MuPDFImageStructuredTextBlock, 59
 Count, 60
 GetEnumerator, 60
 this[int index], 60
 Type, 61
MuPDFCore.MuPDFMultiThreadedPageRenderer, 61
 Abort, 62
 Dispose, 62
 GetProgress, 62
 GetSpanItem, 62
 Render, 63
 ThreadCount, 64
MuPDFCore.MuPDFPage, 64
 Bounds, 65
 Dispose, 65
 PageNumber, 65
MuPDFCore.MuPDFPageCollection, 66
 Count, 67
 Dispose, 67
 GetEnumerator, 67
 Length, 67
 this[int index], 67
MuPDFCore.MuPDFRenderer, 29
MuPDFCore.MuPDFRenderer.PDFRenderer, 93
 Background, 109
 BackgroundProperty, 105
 Contain, 97
 Cover, 97
 Custom, 96
 DisplayArea, 109
 DisplayAreaProperty, 105
 GetProgress, 97
 GetSelectedText, 97
 Highlight, 96
 HighlightBrush, 109
 HighlightBrushProperty, 105
 HighlightedRegions, 109
 HighlightedRegionsProperty, 106
 Initialize, 98–100
 InitializeAsync, 100–103
 IsViewerInitialized, 110
 IsViewerInitializedProperty, 106
 PageBackground, 110
 PageBackgroundProperty, 106
 PageNumber, 110
 PageNumberProperty, 106
 PageSize, 110
 PageSizeProperty, 107
 Pan, 96
 PanHighlight, 96
 PDFRenderer, 97
 PointerEventHandlers, 96
 PointerEventHandlersType, 110
 PointerEventHandlerTypeProperty, 107
 ReleaseResources, 103
 Render, 103
 RenderThreadCount, 111
 RenderThreadCountProperty, 107
 Search, 104
 SelectAll, 104
 Selection, 111
 SelectionBrush, 111
 SelectionBrushProperty, 107
 SelectionProperty, 108
 SetDisplayAreaNow, 104
 Zoom, 111
 ZoomEnabled, 111
 ZoomEnabledProperty, 108
 ZoomIncrement, 112
 ZoomIncrementProperty, 108
 ZoomProperty, 108
 ZoomStep, 105
MuPDFCore.MuPDFRenderer/PDFRenderer.cs, 147
MuPDFCore.MuPDFRenderer/Properties.cs, 167
MuPDFCore.MuPDFRenderer/RectTransition.cs, 171
MuPDFCore.MuPDFStructuredTextAddress, 68
 BlockIndex, 74
 CharacterIndex, 74
 CompareTo, 70
 Equals, 70
 GetHashCode, 71
 Increment, 71
 LineIndex, 74
MuPDFStructuredTextAddress, 69
 operator!=, 71
 operator<, 72
 operator<=, 72
 operator>, 73
 operator>=, 73
 operator==, 72
MuPDFCore.MuPDFStructuredTextAddressSpan, 75
 End, 75
MuPDFStructuredTextAddressSpan, 75
 Start, 76
MuPDFCore.MuPDFStructuredTextBlock, 76
 BoundingBox, 78
 Count, 78
 GetEnumerator, 77
 Image, 77
 Text, 77
 this[int index], 78
 Type, 78
 Types, 77
MuPDFCore.MuPDFStructuredTextCharacter, 79
 BoundingQuad, 80
 Character, 80
 CodePoint, 80
 Color, 80
 Origin, 80
 Size, 80

ToString, 79
MuPDFCore.MuPDFStructuredTextLine, 81
 BoundingBox, 83
 Characters, 83
 Count, 83
 Direction, 84
 GetEnumerator, 82
 Horizontal, 82
 Text, 84
 this[int index], 84
 ToString, 83
 Vertical, 82
 WritingMode, 84
 WritingModes, 82
MuPDFCore.MuPDFStructuredTextPage, 85
 Count, 88
 GetClosestHitAddress, 86
 GetEnumerator, 86
 GetHighlightQuads, 86
 GetHitAddress, 87
 GetText, 87
 Search, 88
 StructuredTextBlocks, 88
 this[int index], 89
 this[MuPDFStructuredTextAddress address], 89
MuPDFCore.MuPDFTextStructuredTextBlock, 90
 Count, 91
 GetEnumerator, 91
 Lines, 91
 this[int index], 91
 ToString, 91
 Type, 92
MuPDFCore.OCRProgressInfo, 92
 Progress, 92
MuPDFCore.PointF, 112
 PointF, 112
 X, 113
 Y, 113
MuPDFCore.Quad, 113
 Contains, 114
 LowerLeft, 115
 LowerRight, 115
 Quad, 114
 UpperLeft, 115
 UpperRight, 115
MuPDFCore.Rectangle, 116
 Contains, 118
 Height, 121
 Intersect, 119
 Rectangle, 117
 Round, 119
 Split, 120
 ToQuad, 120
 Width, 121
 X0, 120
 X1, 121
 Y0, 121
 Y1, 121
 MuPDFCore.RenderProgress, 123
 ThreadRenderProgresses, 123
MuPDFCore.RenderProgress.ThreadRenderProgress,
 144
 MaxProgress, 145
 Progress, 145
MuPDFCore.RoundedRectangle, 124
 Height, 126
 RoundedRectangle, 124
 Split, 125
 Width, 126
 X0, 125
 X1, 125
 Y0, 126
 Y1, 126
MuPDFCore.RoundedSize, 127
 Height, 128
 RoundedSize, 127
 Split, 128
 Width, 128
MuPDFCore.Size, 129
 Height, 130
 Size, 129
 Split, 130
 Width, 130
MuPDFCore.TesseractLanguage, 131
 Afr, 134, 138
 Amh, 134, 138
 Ara, 134, 138
 Arabic, 136, 140
 Armenian, 136, 141
 Asm, 134, 138
 Aze, 134, 138
 Aze_Cyril, 134, 138
 Bel, 134, 138
 Ben, 134, 138
 Bengali, 137, 141
 Best, 133
 BestScripts, 136
 Bod, 134, 138
 Bos, 134, 138
 Bre, 134, 138
 Bul, 134, 138
 Canadian_Aboriginal, 137, 141
 Cat, 134, 138
 Ceb, 134, 138
 Ces, 134, 138
 Cherokee, 137, 141
 Chi_Sim, 134, 138
 Chi_Sim_Vert, 134, 138
 Chi_Tra, 134, 138
 Chi_Tra_Vert, 134, 138
 Chr, 134, 138
 Cos, 134, 138
 Cym, 134, 138
 Cyrillic, 137, 141
 Dan, 134, 138
 Deu, 134, 138

Devanagari, 137, 141
Div, 134, 138
Dzo, 134, 138
Ell, 134, 138
Eng, 134, 138
Enm, 134, 138
Epo, 134, 138
Equ, 138
Est, 134, 138
Ethiopic, 137, 141
Eus, 134, 138
Fao, 134, 138
Fas, 134, 138
Fast, 137
FastScripts, 140
Fil, 134, 138
Fin, 134, 138
Fra, 134, 138
Fraktur, 137, 141
Frk, 134, 138
Frm, 134, 138
Fry, 134, 138
Georgian, 137, 141
Gla, 134, 138
Gle, 134, 138
Glg, 134, 138
Grc, 134, 138
Greek, 137, 141
Guj, 134, 138
Gujarati, 137, 141
Gurmukhi, 137, 141
Hangul, 137, 141
Hangul_Vert, 137, 141
HanS, 137, 141
HanS_Vert, 137, 141
HanT, 137, 141
HanT_Vert, 137, 141
Hat, 134, 139
Heb, 135, 139
Hebrew, 137, 141
Hin, 135, 139
Hrv, 135, 139
Hun, 135, 139
Hye, 135, 139
Iku, 135, 139
Ind, 135, 139
Isl, 135, 139
Ita, 135, 139
Ita_Old, 135, 139
Japanese, 137, 141
Japanese_Vert, 137, 141
Jav, 135, 139
Jpn, 135, 139
Jpn_Vert, 135, 139
Kan, 135, 139
Kannada, 137, 141
Kat, 135, 139
Kat_Old, 135, 139
Kaz, 135, 139
Khm, 135, 139
Khmer, 137, 141
Kir, 135, 139
Kmr, 135, 139
Kor, 135, 139
Kor_Vert, 135, 139
Language, 144
Lao, 135, 137, 139, 141
Lat, 135, 139
Latin, 137, 141
Lav, 135, 139
Lit, 135, 139
Ltz, 135, 139
Mal, 135, 139
Malayalam, 137, 141
Mar, 135, 139
Mkd, 135, 139
Mlt, 135, 139
Mon, 135, 139
Mri, 135, 139
Msa, 135, 139
Mya, 135, 139
Myanmar, 137, 141
Nep, 135, 139
Nld, 135, 139
Nor, 135, 139
Oci, 135, 139
Ori, 135, 139
Oriya, 137, 141
Osd, 135, 139
Pan, 135, 139
Pol, 135, 139
Por, 135, 139
Prefix, 144
Pus, 135, 139
Que, 135, 140
Ron, 136, 140
Rus, 136, 140
San, 136, 140
Sin, 136, 140
Sinhala, 137, 141
Slk, 136, 140
Slv, 136, 140
Snd, 136, 140
Spa, 136, 140
Spa_Old, 136, 140
Sqj, 136, 140
Srp, 136, 140
Srp_Latn, 136, 140
Sun, 136, 140
Swa, 136, 140
Swe, 136, 140
Syr, 136, 140
Syriac, 137, 141
Tam, 136, 140
Tamil, 137, 141
Tat, 136, 140

Tel, 136, 140
Telugu, 137, 141
TesseractLanguage, 141–143
Tgk, 136, 140
Tha, 136, 140
Thaana, 137, 141
Thai, 137, 141
Tibetan, 137, 141
Tir, 136, 140
Ton, 136, 140
Tur, 136, 140
Uig, 136, 140
Ukr, 136, 140
Urd, 136, 140
Uzb, 136, 140
Uzb_Cyril, 136, 140
Vie, 136, 140
Vietnamese, 137, 141
Yid, 136, 140
Yor, 136, 140
MuPDFCore/MuPDF.cs, 172
MuPDFCore/MuPDFContext.cs, 191
MuPDFCore/MuPDFDisplayList.cs, 193
MuPDFCore/MuPDFDocument.cs, 194
MuPDFCore/MuPDFMultiThreadedPageRenderer.cs, 214
MuPDFCore/MuPDFPage.cs, 222
MuPDFCore/MuPDFStructuredTextPage.cs, 224
MuPDFCore/Rectangles.cs, 241
MuPDFCore/TesseractLanguage.cs, 249
MuPDFCore/Utils.cs, 269
MuPDFDocument
 MuPDFCore.MuPDFDocument, 41–43
MuPDFStructuredTextAddress
 MuPDFCore.MuPDFStructuredTextAddress, 69
MuPDFStructuredTextAddressSpan
 MuPDFCore.MuPDFStructuredTextAddressSpan, 75
Mya
 MuPDFCore.TesseractLanguage, 135, 139
Myanmar
 MuPDFCore.TesseractLanguage, 137, 141
Nep
 MuPDFCore.TesseractLanguage, 135, 139
Nld
 MuPDFCore.TesseractLanguage, 135, 139
None
 MuPDFCore, 26, 27
Nor
 MuPDFCore.TesseractLanguage, 135, 139
Oci
 MuPDFCore.TesseractLanguage, 135, 139
operator!=
 MuPDFCore.MuPDFStructuredTextAddress, 71
operator<
 MuPDFCore.MuPDFStructuredTextAddress, 72
operator<=

MuPDFCore.MuPDFStructuredTextAddress, 72
operator>
 MuPDFCore.MuPDFStructuredTextAddress, 73
operator>=
 MuPDFCore.MuPDFStructuredTextAddress, 73
operator==
 MuPDFCore.MuPDFStructuredTextAddress, 72
Ori
 MuPDFCore.TesseractLanguage, 135, 139
Origin
 MuPDFCore.MuPDFStructuredTextCharacter, 80
Oriya
 MuPDFCore.TesseractLanguage, 137, 141
Osd
 MuPDFCore.TesseractLanguage, 135, 139
Owner
 MuPDFCore, 27
PageBackground
 MuPDFCore.MuPDFRenderer.PDFRenderer, 110
PageBackgroundProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 106
PageNumber
 MuPDFCore.MuPDFPage, 65
 MuPDFCore.MuPDFRenderer.PDFRenderer, 110
PageNumberProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 106
Pages
 MuPDFCore.MuPDFDocument, 57
PageSize
 MuPDFCore.MuPDFRenderer.PDFRenderer, 110
PageSizeProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 107
PAM
 MuPDFCore, 27, 28
Pan
 MuPDFCore.MuPDFRenderer.PDFRenderer, 96
 MuPDFCore.TesseractLanguage, 135, 139
PanHighlight
 MuPDFCore.MuPDFRenderer.PDFRenderer, 96
PasswordTypes
 MuPDFCore, 27
PDF
 MuPDFCore, 25, 27
PDFRenderer
 MuPDFCore.MuPDFRenderer.PDFRenderer, 97
PixelFormats
 MuPDFCore, 28
PNG
 MuPDFCore, 27, 28
PNM
 MuPDFCore, 27, 28
PointerEventHandlers
 MuPDFCore.MuPDFRenderer.PDFRenderer, 96
PointerEventHandlersType
 MuPDFCore.MuPDFRenderer.PDFRenderer, 110
PointerEventHandlerTypeProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 107
PointF

MuPDFCore.PointF, 112
Pol
 MuPDFCore.TesseractLanguage, 135, 139
Por
 MuPDFCore.TesseractLanguage, 135, 139
Prefix
 MuPDFCore.TesseractLanguage, 144
Print
 MuPDFCore, 26
Progress
 MuPDFCore.OCRProgressInfo, 92
 MuPDFCore.RenderProgress.ThreadRenderProgress, 145
PSD
 MuPDFCore, 28
Pus
 MuPDFCore.TesseractLanguage, 135, 139
Quad
 MuPDFCore.Quad, 114
Que
 MuPDFCore.TesseractLanguage, 135, 140
RasterOutputFileTypes
 MuPDFCore, 28
Rectangle
 MuPDFCore.Rectangle, 117
RedirectOutput
 MuPDFCore.MuPDF, 35
ReleaseResources
 MuPDFCore.MuPDFRenderer.PDFRenderer, 103
Render
 MuPDFCore.MuPDFDocument, 51–54
 MuPDFCore.MuPDFMultiThreadedPageRenderer, 63
 MuPDFCore.MuPDFRenderer.PDFRenderer, 103
RenderThreadCount
 MuPDFCore.MuPDFRenderer.PDFRenderer, 111
RenderThreadCountProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 107
ResetOutput
 MuPDFCore.MuPDF, 35
Restricted
 MuPDFCore, 28
Restrictions
 MuPDFCore.MuPDFDocument, 58
RestrictionState
 MuPDFCore, 28
 MuPDFCore.MuPDFDocument, 58
RGB
 MuPDFCore, 28
RGBA
 MuPDFCore, 28
Ron
 MuPDFCore.TesseractLanguage, 136, 140
Round
 MuPDFCore.Rectangle, 119
RoundedRectangle
 MuPDFCore.RoundedRectangle, 124
RoundedSize
 MuPDFCore.RoundedSize, 127
Rus
 MuPDFCore.TesseractLanguage, 136, 140
San
 MuPDFCore.TesseractLanguage, 136, 140
SavImage
 MuPDFCore.MuPDFDocument, 54, 55
Search
 MuPDFCore.MuPDFRenderer.PDFRenderer, 104
 MuPDFCore.MuPDFStructuredTextPage, 88
SelectAll
 MuPDFCore.MuPDFRenderer.PDFRenderer, 104
Selection
 MuPDFCore.MuPDFRenderer.PDFRenderer, 111
SelectionBrush
 MuPDFCore.MuPDFRenderer.PDFRenderer, 111
SelectionBrushProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 107
SelectionProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 108
SetDisplayAreaNow
 MuPDFCore.MuPDFRenderer.PDFRenderer, 104
ShrinkStore
 MuPDFCore.MuPDFContext, 38
Sin
 MuPDFCore.TesseractLanguage, 136, 140
Sinhala
 MuPDFCore.TesseractLanguage, 137, 141
Size
 MuPDFCore.MuPDFStructuredTextCharacter, 80
 MuPDFCore.Size, 129
Slk
 MuPDFCore.TesseractLanguage, 136, 140
Slv
 MuPDFCore.TesseractLanguage, 136, 140
Snd
 MuPDFCore.TesseractLanguage, 136, 140
Spa
 MuPDFCore.TesseractLanguage, 136, 140
Spa_Old
 MuPDFCore.TesseractLanguage, 136, 140
Split
 MuPDFCore.Rectangle, 120
 MuPDFCore.RoundedRectangle, 125
 MuPDFCore.RoundedSize, 128
 MuPDFCore.Size, 130
Sqi
 MuPDFCore.TesseractLanguage, 136, 140
Srp
 MuPDFCore.TesseractLanguage, 136, 140
Srp_Latn
 MuPDFCore.TesseractLanguage, 136, 140
StandardErrorMessage
 MuPDFCore.MuPDF, 36
StandardOutputMessage
 MuPDFCore.MuPDF, 36
Start

MuPDFCore.MuPDFStructuredTextAddressSpan, 76
StoreMaxSize MuPDFCore.MuPDFContext, 38
StoreSize MuPDFCore.MuPDFContext, 38
StructuredTextBlocks MuPDFCore.MuPDFStructuredTextPage, 88
Sun MuPDFCore.TesseractLanguage, 136, 140
SVG MuPDFCore, 25
Swa MuPDFCore.TesseractLanguage, 136, 140
Swe MuPDFCore.TesseractLanguage, 136, 140
Syr MuPDFCore.TesseractLanguage, 136, 140
Syriac MuPDFCore.TesseractLanguage, 137, 141
Tam MuPDFCore.TesseractLanguage, 136, 140
Tamil MuPDFCore.TesseractLanguage, 137, 141
Tat MuPDFCore.TesseractLanguage, 136, 140
Tel MuPDFCore.TesseractLanguage, 136, 140
Telugu MuPDFCore.TesseractLanguage, 137, 141
TesseractLanguage MuPDFCore.TesseractLanguage, 141–143
Text MuPDFCore.MuPDFStructuredTextBlock, 77
MuPDFCore.MuPDFStructuredTextLine, 84
Tgk MuPDFCore.TesseractLanguage, 136, 140
Tha MuPDFCore.TesseractLanguage, 136, 140
Thaana MuPDFCore.TesseractLanguage, 137, 141
Thai MuPDFCore.TesseractLanguage, 137, 141
this[int index] MuPDFCore.MuPDFImageStructuredTextBlock, 60
MuPDFCore.MuPDFPageCollection, 67
MuPDFCore.MuPDFStructuredTextBlock, 78
MuPDFCore.MuPDFStructuredTextLine, 84
MuPDFCore.MuPDFStructuredTextPage, 89
MuPDFCore.MuPDFTextStructuredTextBlock, 91
this[MuPDFStructuredTextAddress address] MuPDFCore.MuPDFStructuredTextPage, 89
ThreadCount MuPDFCore.MuPDFMultiThreadedPageRenderer, 64
ThreadRenderProgresses MuPDFCore.RenderProgress, 123
Tibetan
MuPDFCore.TesseractLanguage, 137, 141
TIFF MuPDFCore, 27
Tir MuPDFCore.TesseractLanguage, 136, 140
Ton MuPDFCore.TesseractLanguage, 136, 140
ToQuad MuPDFCore.Rectangle, 120
ToString MuPDFCore.MuPDFStructuredTextCharacter, 79
MuPDFCore.MuPDFStructuredTextLine, 83
MuPDFCore.MuPDFTextStructuredTextBlock, 91
TryUnlock MuPDFCore.MuPDFDocument, 55, 56
Tur MuPDFCore.TesseractLanguage, 136, 140
Type MuPDFCore.MuPDFImageStructuredTextBlock, 61
MuPDFCore.MuPDFStructuredTextBlock, 78
MuPDFCore.MuPDFTextStructuredTextBlock, 92
Types MuPDFCore.MuPDFStructuredTextBlock, 77
Uig MuPDFCore.TesseractLanguage, 136, 140
Ukr MuPDFCore.TesseractLanguage, 136, 140
Unencrypted MuPDFCore, 26
Unlocked MuPDFCore, 26, 28
Unrestricted MuPDFCore, 28
UpperLeft MuPDFCore.Quad, 115
UpperRight MuPDFCore.Quad, 115
Urd MuPDFCore.TesseractLanguage, 136, 140
User MuPDFCore, 27
Uzb MuPDFCore.TesseractLanguage, 136, 140
Uzb_Cyril MuPDFCore.TesseractLanguage, 136, 140
Vertical MuPDFCore.MuPDFStructuredTextLine, 82
Vie MuPDFCore.TesseractLanguage, 136, 140
Vietnamese MuPDFCore.TesseractLanguage, 137, 141
Width MuPDFCore.Rectangle, 121
MuPDFCore.RoundedRectangle, 126
MuPDFCore.RoundedSize, 128
MuPDFCore.Size, 130

WriteImage
 MuPDFCore.MuPDFDocument, 56, 57

WritingMode
 MuPDFCore.MuPDFStructuredTextLine, 84

WritingModes
 MuPDFCore.MuPDFStructuredTextLine, 82

X
 MuPDFCore.PointF, 113

X0
 MuPDFCore.Rectangle, 120
 MuPDFCore.RoundedRectangle, 125

X1
 MuPDFCore.Rectangle, 121
 MuPDFCore.RoundedRectangle, 125

XPS
 MuPDFCore, 27

Y
 MuPDFCore.PointF, 113

Y0
 MuPDFCore.Rectangle, 121
 MuPDFCore.RoundedRectangle, 126

Y1
 MuPDFCore.Rectangle, 121
 MuPDFCore.RoundedRectangle, 126

Yid
 MuPDFCore.TesseractLanguage, 136, 140

Yor
 MuPDFCore.TesseractLanguage, 136, 140

Zoom
 MuPDFCore.MuPDFRenderer.PDFRenderer, 111

ZoomEnabled
 MuPDFCore.MuPDFRenderer.PDFRenderer, 111

ZoomEnabledProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 108

ZoomIncrement
 MuPDFCore.MuPDFRenderer.PDFRenderer, 112

ZoomIncrementProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 108

ZoomProperty
 MuPDFCore.MuPDFRenderer.PDFRenderer, 108

ZoomStep
 MuPDFCore.MuPDFRenderer.PDFRenderer, 105